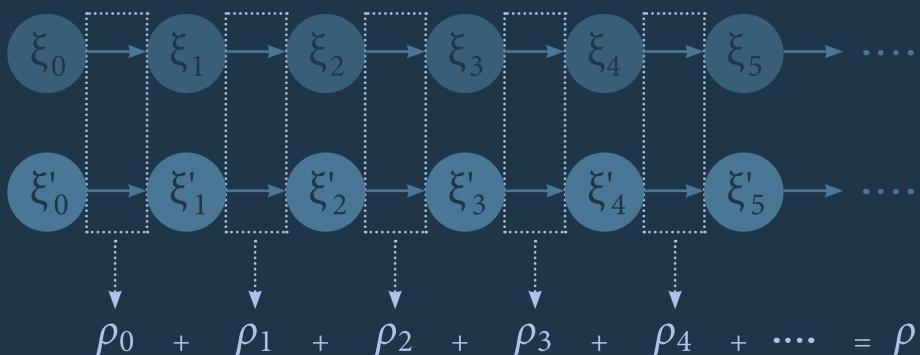


Zhi-Hua Zhou · Yang Yu · Chao Qian

Evolutionary Learning

Advances
in Theories
and Algorithms



Evolutionary Learning: Advances in Theories and Algorithms

Zhi-Hua Zhou · Yang Yu ·
Chao Qian

Evolutionary Learning: Advances in Theories and Algorithms



Springer

Zhi-Hua Zhou
Nanjing University
Nanjing, Jiangsu, China

Yang Yu
Nanjing University
Nanjing, Jiangsu, China

Chao Qian
Nanjing University
Nanjing, Jiangsu, China

ISBN 978-981-13-5955-2 ISBN 978-981-13-5956-9 (eBook)
<https://doi.org/10.1007/978-981-13-5956-9>

© Springer Nature Singapore Pte Ltd. 2019

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721,
Singapore

Preface

Around the year of 2001, the first author of this monograph, Zhou, developed with his collaborators a powerful selective ensemble approach. This approach is able to produce small-sized ensembles with generalization performance even stronger than that of full-sized ones by exploiting genetic algorithm, a popularly used evolutionary algorithm (EA). Zhou realized that EAs are powerful optimization techniques that can be well useful in various machine learning tasks. At that time, however, EAs were almost purely heuristic, not favored by the machine learning community with strong theoretical flavor. Impressed by the successes of EAs in applications, Zhou believed there must be some theoretical explanations behind their mysteries and decided to start this track of research. In 2004, the second author of this monograph, Yu, finished his bachelor thesis on selective ensemble under the supervision of Zhou. Yu then joined Zhou as a PhD student taking theoretical aspects of EAs as his thesis topic, and obtained PhD degree in 2011. In 2009, Zhou accepted the third author of this monograph, Qian, as his PhD student with theories and algorithms of evolutionary learning for his thesis topic, and Qian obtained his PhD degree in 2015. Overall, the majority of contents in this monograph are research results achieved by the authors during the past two decades.

This monograph is composed of four parts. Part I briefly introduces evolutionary learning and some preliminaries. Aiming at analyzing the running time complexity and the approximation ability, the two essential theoretical aspects of EAs, Part II presents two general approaches for deriving running time bounds and a general framework for characterizing the approximation performance. These results serve as general tools for attaining many theoretical results reported in the remainder of the monograph. Part III presents a series of theoretical studies, mainly about the influence of major factors, such as recombination, solution representation, inaccurate fitness evaluation, population, etc., on the performance of EAs. In Part IV, the authors come back to selective ensemble, their original motivating task for studying EAs, and present algorithms behaving at least as well as state-of-the-art

selective ensemble algorithms. The authors then study subset selection, a more general problem widely occurring in various machine learning tasks, and present a series of evolutionary learning algorithms with bounded approximation guarantees.

The authors hope that the general theoretical tools presented in Part II can be found useful for interested readers to explore theoretical foundations of evolutionary learning, the theoretical results presented in Part III can help readers to get more understanding about behaviors of evolutionary learning processes and offer some insight for algorithm design, and the algorithms presented in Part IV can be found useful in real-world applications of machine learning.

The authors want to thank their families, friends and collaborators. Gratitude goes to Celine Chang and Alfred Hofmann who encouraged authors to publish this monograph with Springer.

*The authors
February 2019*

Notations

\mathbb{R}	real number
\mathbb{N}	integer
$(\cdot)^+$	positive, (\cdot) can be \mathbb{R} or \mathbb{N}
$(\cdot)^{0+}$	non-negative, (\cdot) can be \mathbb{R} or \mathbb{N}
x	variable
x	vector
$(\cdot, \cdot, \dots, \cdot)$	row vector
$0, 1$	all-0s and all-1s vectors
$ \cdot _0, \cdot _1$	number of 0s and 1s of a vector
$\{0, 1\}^n$	Boolean vector space
X	matrix
$(\cdot)^T$	transpose of a vector/matrix
X	set
$\{\cdot, \cdot, \dots, \cdot\}$	set by enumeration
$[n]$	set $\{1, 2, \dots, n\}$
$ \cdot $	cardinality of a set
2^X	power set of X , which consists of all subsets of X
$X \setminus Y$	complement of Y in X , which consists of elements in X but not in Y

VIII Notations

x	state of a Markov chain
\mathcal{X}	state space of a Markov chain
$P(\cdot), P(\cdot \cdot)$	probability and conditional probability
π	probability distribution
f	function
$\mathbb{E}_{\sim \pi}[f(\cdot)], \mathbb{E}_{\sim \pi}[f(\cdot) \cdot]$	expectation and conditional expectation of $f(\cdot)$ under distribution π , simplified as $\mathbb{E}[f(\cdot)], \mathbb{E}[f(\cdot) \cdot]$ when the meaning is clear
$\mathbb{I}(\cdot)$	indicator function which takes 1 if \cdot is true, and 0 otherwise
$\lfloor \cdot \rfloor, \lceil \cdot \rceil$	floor and ceiling functions which take the greatest/least integer less/greater than or equal to a real number
OPT	optimal function value
H_n	n -th harmonic number, i.e., $\sum_{i=1}^n (1/i)$
\forall	for all
\exists, \nexists	there exists, does not exist

Contents

Part I INTRODUCTION

1	Introduction	3
1.1	Machine Learning	3
1.2	Evolutionary Learning	4
1.3	Multi-objective Optimization	6
1.4	Organization of the Book	9
2	Preliminaries	11
2.1	Evolutionary Algorithms	11
2.2	Pseudo-Boolean Functions	15
2.3	Running Time Complexity	17
2.4	Markov Chain Modeling	19
2.5	Analysis Tools	21

Part II ANALYSIS METHODOLOGY

3	Running Time Analysis: Convergence-based Analysis	29
3.1	Convergence-based Analysis: Framework	30
3.2	Convergence-based Analysis: Application Illustration	34
3.3	Summary	39
4	Running Time Analysis: Switch Analysis	41
4.1	Switch Analysis: Framework	41
4.2	Switch Analysis: Application Illustration	46
4.3	Summary	50
5	Running Time Analysis: Comparison and Unification	51
5.1	Analysis Approaches: Formalization	51
5.2	Switch Analysis vs. Fitness Level	53

5.3	Switch Analysis vs. Drift Analysis	57
5.4	Switch Analysis vs. Convergence-based Analysis.....	61
5.5	Analysis Approaches: Unification	65
5.6	Summary	67
6	Approximation Analysis: SEIP	69
6.1	SEIP: Framework	70
6.2	SEIP: Application Illustration	77
6.3	Summary	80

Part III THEORETICAL PERSPECTIVES

7	Boundary Problems of EAs	83
7.1	Boundary Problem Identification	84
7.2	Case Study	87
7.3	Summary	92
8	Recombination	93
8.1	Recombination and Mutation	95
8.2	Recombination Enabled MOEAs	97
8.3	Case Study	100
8.4	Empirical Verification	106
8.5	Summary	108
9	Representation	109
9.1	Genetic Programming Representation	111
9.2	Case Study: Maximum Matchings	113
9.3	Case Study: Minimum Spanning Trees	119
9.4	Empirical Verification	125
9.5	Summary	128
10	Inaccurate Fitness Evaluation	129
10.1	Noisy Optimization	130
10.2	Influence of Noisy Fitness	132
10.3	Denoise by Threshold Selection	136
10.4	Denoise by Sampling	141
10.5	Empirical Verification	148
10.6	Summary	152
11	Population	155
11.1	Influence of Population	156
11.2	Robustness of Population against Noise	160
11.3	Summary	172

12 Constrained Optimization	175
12.1 Usefulness of Infeasible Solutions	177
12.2 Effectiveness of Pareto Optimization	183
12.3 Summary	194

Part IV LEARNING ALGORITHMS

13 Selective Ensemble	197
13.1 Selective Ensemble	198
13.2 The POSE Algorithm	200
13.3 Theoretical Analysis	202
13.4 Empirical Study	208
13.5 Summary	214
14 Subset Selection	215
14.1 Subset Selection	216
14.2 The POSS Algorithm	221
14.3 Theoretical Analysis	222
14.4 Empirical Study	226
14.5 Summary	230
15 Subset Selection: k-Submodular Maximization	233
15.1 Monotone k -Submodular Function Maximization	235
15.2 The POkSS Algorithm	238
15.3 Theoretical Analysis	241
15.4 Empirical Study	246
15.5 Summary	254
16 Subset Selection: Ratio Minimization	255
16.1 Ratio Minimization of Monotone Submodular Functions	256
16.2 The PORM Algorithm	259
16.3 Theoretical Analysis	261
16.4 Empirical Study	266
16.5 Summary	268
17 Subset Selection: Noise	269
17.1 Noisy Subset Selection	270
17.2 The PONSS Algorithm	275
17.3 Theoretical Analysis	277
17.4 Empirical Study	280
17.5 Summary	283

18	Subset Selection: Acceleration	285
18.1	The PPOSS Algorithm	285
18.2	Theoretical Analysis	287
18.3	Empirical Study	291
18.4	Summary	293
Appendix		295
A.1	Proofs in Chapter 4	295
A.2	Proofs in Chapter 5	298
A.3	Proofs in Chapter 8	306
A.4	Proofs in Chapter 9	310
A.5	Proofs in Chapter 10	312
A.6	Proofs in Chapter 11	334
A.7	Proofs in Chapter 12	335
A.8	Proofs in Chapter 15	339
A.9	Proofs in Chapter 17	340
References		343

Part I

INTRODUCTION



1

Introduction

1.1 Machine Learning

Machine learning [Mitchell, 1997] is a central topic of artificial intelligence. It aims at learning generalizable *models* from data, with which the system performance can be improved. As nowadays data can be continually acquired and accumulated in numerous applications such as image recognition and commodity recommendation, machine learning has been playing an increasingly important role in the successes of these applications.

To learn a model, a set of instances (called a *training set*) is required. If each instance is given with a target output, the learning falls into the type of *supervised learning*, which is to learn a model mapping from the instance space to the target space. According to the type of the target variable, supervised learning tasks can be roughly categorized into *classification* and *regression* tasks. The target variable takes a finite number of categories in classification and is continuous in regression. Typical supervised learning algorithms include *decision trees* [Quinlan, 1993], *artificial neural networks* [Haykin, 1998], *support vector machines* [Cristianini and Shawe-Taylor, 2000], etc. If no instance is given with any target output, it falls into the type of *unsupervised learning*. A typical task is *clustering* [Jain and Dubes, 1988], which is to group instances into clusters such that the instances belonging to the same cluster are similar while those belonging to different clusters are dissimilar. *Weakly supervised learning* [Zhou, 2017] is somewhere between supervised and unsupervised learning, including *semi-supervised learning* [Zhu, 2008] and *active learning* [Settles, 2010] with incomplete supervision, *multi-instance learning* [Dietterich et al., 1997] with inexact supervision, *reinforcement learning* [Sutton and Barto, 1998] with time-delayed supervision, etc.

One perspective is to view every machine learning algorithm as a composition of three components [Domingos, 2012]: *model representation*, *evaluation* and *optimization*, as shown in Figure 1.1. The model to be learned can be represented in different ways, e.g., decision trees, artificial neural

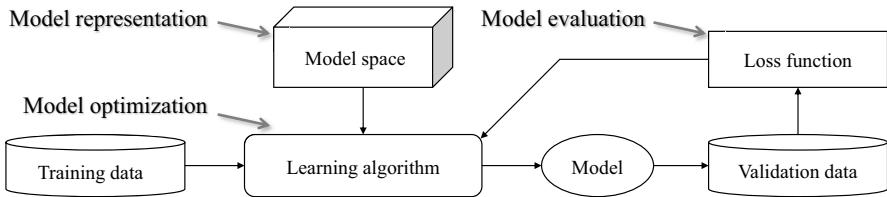


Figure 1.1. The three components in a typical machine learning process.

networks and support vector machines. Different model representations have different characteristics, which can largely influence the generalization performance of the model. Model evaluation is about how to measure the goodness of a model. This is crucial because, on the one hand, the same model may exhibit different goodness according to different evaluation criteria; on the other hand, machine learning models are with rich variations, e.g., parameter values, and a proper selection relies on effective model evaluation. Model optimization is about how to determine the parameter values to enable the model to have good generalization performance.

To solve complicated learning tasks, one often needs to employ non-linear model representation and/or non-convex model evaluation functions. These commonly result in a learning problem formulated as a complicated optimization problem. The objective function to be optimized can be non-differentiable, non-continuous, and have many local optima. These properties may make conventional optimization algorithms such as gradient decent fail, while other powerful optimization algorithms, such as those to be discussed in this book, may be found well helpful.

1.2 Evolutionary Learning

Evolutionary algorithms (EAs) [Bäck, 1996] are a large class of heuristic randomized optimization algorithms, inspired by natural evolution. They simulate the natural evolution process by considering two key factors, i.e., variational reproduction and superior selection. Despite many different implementations, e.g., *genetic algorithm* (GA), *genetic programming* (GP) and *evolutionary strategy* (ES), typical EAs can be abstracted as the following four steps:

1. Generate an initial set of solutions (called a *population*);
2. Reproduce new solutions based on the current population;
3. Remove relatively poor solutions in the population;
4. Repeat from Step 2 until some stop criterion is met.

The steps are embodied in the general structure of EAs shown in Figure 1.2. Before using EAs to solve an optimization problem, one needs to

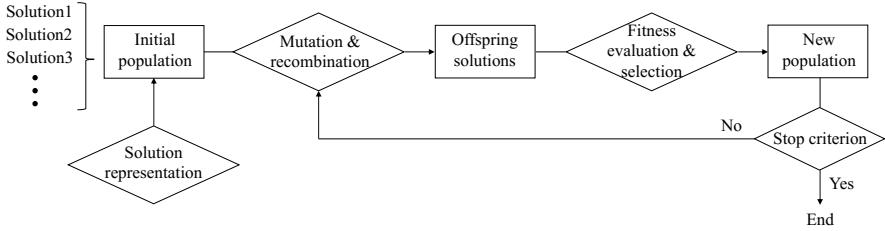


Figure 1.2. The general structure of EAs.

decide how to **represent a solution**. For example, if the problem is to select a subset from a ground set, a subset corresponding to a solution can be naturally represented by a Boolean vector. As shown in Figure 1.3, a subset of $\{v_1, v_2, \dots, v_8\}$ can be represented by a Boolean vector of length 8, where the i -th entry is 1 means that v_i is selected, otherwise v_i is not selected; thus $\{v_1, v_3, v_4, v_5\}$ can be represented by $(1, 0, 1, 1, 1, 0, 0, 0)$.

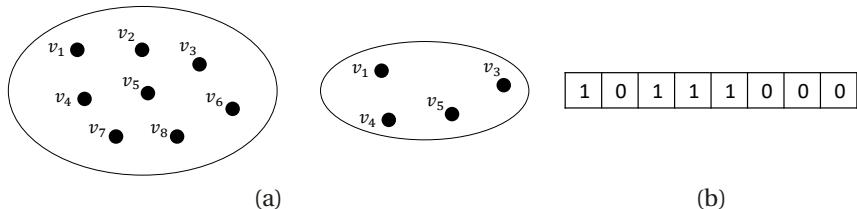


Figure 1.3. An example of solution representation: (a) a ground set consisting of 8 elements and a subset containing $\{v_1, v_3, v_4, v_5\}$ (b) the binary vector representation of the subset $\{v_1, v_3, v_4, v_5\}$.

Based on the solution representation, EAs step into the evolutionary cycle, by the loop in Figure 1.2. In this cycle, EAs maintain a population of solutions and improve the population by iteratively reproducing new offspring solutions, evaluating them and selecting better solutions. *Mutation* and *recombination* (or called *crossover*) are two popular operators for reproduction. Mutation randomly changes one solution to generate a new one. Figure 1.4 gives an example of one-bit mutation over a Boolean vector solution, which flips a randomly chosen bit. Recombination mixes up two or more solutions to generate new ones. Figure 1.5 gives an example of one-point recombination over two Boolean vector solutions, which exchanges the parts of the two solutions separated by a randomly chosen position.

A *fitness* function is used to measure the goodness of newly generated offspring solutions. Good solutions from the old population and the offspring solutions are selected to form the new population according to some selection mechanism. The evolutionary cycle stops when the stop criterion



Figure 1.4. One-bit mutation over a Boolean vector solution. For the parent solution, a position is chosen randomly, and one offspring solution is generated by flipping the bit at the position of the parent solution.

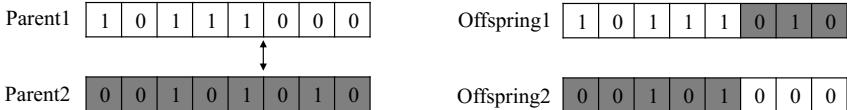


Figure 1.5. One-point recombination over two Boolean vector solutions. For the two parent solutions, a position is chosen randomly, and two offspring solutions are generated by exchanging the parts of the parent solutions after the position.

is met. There can be several criteria, often including that a predefined solution quality has been met, the computational resource (e.g., running time) budget runs out, or the solutions have no longer been improved for predefined iterations.

From the procedure of EAs, one can see that, for solving optimization problems, EAs only require the solutions to be represented and evaluated in order to perform the search, while problem structure information can be unavailable. Particularly, EAs can be applied without gradient information of the objective function or even explicit objective function formulation; they only require the goodness of a solution to be evaluated by experiments or simulations. Thus, EAs are regarded as general-purpose optimization algorithms, and can even be applied in the “black-box” manner to solve optimization problems.

Due to the general-purpose property, EAs have been applied to solve complicated optimization problems in machine learning. For instance, EAs have been used to optimize artificial neural networks [Yao, 1999], including optimizing connection weights, architectures and learning rules; the evolved artificial neural network models can achieve competitive performance to the hand-designed models [Real et al., 2017]. However, though *evolutionary learning* has achieved successes, like these studies, the lack of sound theoretical foundation encumbers its acceptance to the general machine learning community.

1.3 Multi-objective Optimization

In many machine learning tasks, the expected outcome naturally has multiple desired properties. For instances, *feature selection* [Liu and Motoda, 1998] tries to optimize some performance measures using as few features

as possible; *ensemble learning* [Zhou, 2012] desires the individual learners to be accurate and diverse at the same time; clustering [Jain and Dubes, 1988] requires minimizing the distance of instances within the same cluster but maximizing the distance of instances between different clusters; active learning [Settles, 2010] desires the selected unlabeled instances to be informative, representative and diverse [Huang et al., 2014]. An easy way to satisfy multiple properties is to sum them up, with balancing weights, in a single objective function. However, proper weights are not often easy to be determined a priori. Alternatively, when we do not know how to balance the properties, but can only treat each of them as a separate objective function, we will be facing *multi-objective* optimization, which is quite different from *single-objective* optimization. In the following, we will introduce some basic concepts.

Multi-objective optimization requires to simultaneously optimize two or more objective functions, without explicitly balancing them, as shown in Definition 1.1. A solution subspace which accommodates only and all *feasible* solutions is called a feasible solution space. When there are two objective functions, it is also called *bi-objective* optimization. In this book, maximization is considered, as minimization can be defined similarly.

Definition 1.1 (Multi-objective Optimization). *Given a feasible solution space S and m objective functions f_1, f_2, \dots, f_m , the multi objective optimization (for maximization) aims to find the solution s^* satisfying*

$$s^* = \arg \max_{s \in S} f(s) = \arg \max_{s \in S} (f_1(s), f_2(s), \dots, f_m(s)), \quad (1.1)$$

where $f(s) = (f_1(s), f_2(s), \dots, f_m(s))$ is the objective vector of a solution s .

The objective functions are usually conflicted with each other, i.e., optimization of one objective alone may degrade other objectives, and therefore, it is impossible to have one solution which is optimal for all objectives. Thus, the outcome of multi-objective optimization is a set of solutions, rather than a single one. Each solution in this set has some unique advantage, compared with all other solutions. The advantage can be measured by some criteria. A commonly used one is the *Pareto optimality* in Definition 1.3, which utilizes the *domination* relation between solutions as in Definition 1.2.

Definition 1.2 (Domination). *Given a feasible solution space S and an objective space \mathbb{R}^m , let $f = (f_1, f_2, \dots, f_m) : S \rightarrow \mathbb{R}^m$ be the objective vector. For two solutions s and $s' \in S$:*

1. s weakly dominates s' , denoted as $s \succeq s'$, if $\forall i \in [m] : f_i(s) \geq f_i(s')$;
2. s dominates s' , denoted as $s \succ s'$, if $s \succeq s'$ and $\exists i \in [m] : f_i(s) > f_i(s')$.

Two solutions s and s' are *incomparable*, if neither $s \succeq s'$ nor $s' \succeq s$. Note that the above definition is for multi-objective maximization. One can

define the domination relation similarly for multi-objective minimization by using notations $s \preceq s'$ and $s \prec s'$, respectively.

Definition 1.3 (Pareto Optimality). *Given a feasible solution space S and an objective space \mathbb{R}^m , let $f = (f_1, f_2, \dots, f_m) : S \rightarrow \mathbb{R}^m$ be the objective vector. A solution is Pareto optimal if no other solution in S dominates it. The collection of objective vectors of all Pareto optimal solutions is called the Pareto front.*

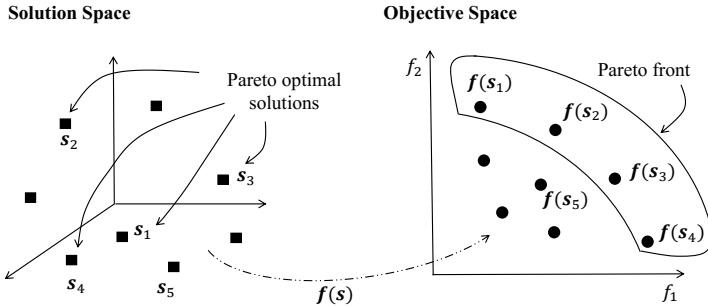


Figure 1.6. Illustration of Pareto optimal solutions and Pareto front.

Figure 1.6 gives an illustration of bi-objective maximization, where f_1 and f_2 are two objectives to be maximized. Black squares and dots represent solutions and their objective vectors, respectively, and the feasible solution space contains totally 8 solutions. It can be seen that the solution $s_2 \succeq s_5$ as $f_1(s_2) \geq f_1(s_5)$ and $f_2(s_2) \geq f_2(s_5)$. Furthermore, $s_2 \succ s_5$ because it actually holds that $f_1(s_2) > f_1(s_5)$ and $f_2(s_2) > f_2(s_5)$. For s_2 and s_3 , however, $f_1(s_2) < f_1(s_3)$ but $f_2(s_2) > f_2(s_3)$; thus they are incomparable. Among these 8 solutions, s_1 , s_2 , s_3 and s_4 are Pareto optimal; their corresponding objective vectors $f(s_1)$, $f(s_2)$, $f(s_3)$ and $f(s_4)$ constitute the Pareto front. It is evident that Pareto optimal solutions with different objective vectors attain optimal trade-offs between multiple objective functions in different ways.

Therefore, towards the Pareto optimality, the goal of multi-objective optimization is to find a set of solutions whose objective vectors cover the Pareto front.

To solve multi-objective optimization problems, a classical way is to transform the multi-objective optimization problem into a single-objective optimization problem by assigning the balancing weights and summing up the multiple objective functions. However, this kind of methods find only one solution each time, and the weights for finding different Pareto optimal solutions would be quite tricky to be assigned.

EAs, as we have described, are population-based, i.e., they maintain a set of solutions. The population-based nature matches the solution set requirement of multi-objective optimization. EAs can be adopted to find the set of Pareto optimal solutions at once, and thus have become a popular tool for this purpose [Deb, 2001]. EAs for multi-objective optimization are called multi-objective EAs (MOEAs). MOEAs have been applied to tackle the multi-objective optimization problems in various machine learning tasks, including feature selection [Mukhopadhyay et al., 2014a], ensemble learning [Chen and Yao, 2010], clustering [Mukhopadhyay et al., 2014b], active learning [Reyes and Ventura, 2018] and *multi-label learning* [Shi et al., 2014]. However, the applicability of EAs does not automatically provide the soundness of the optimization. On the contrary, empirical observations have been keeping raising questions about the understanding of problems and algorithms, which have to be answered theoretically. This motivates our studies reported in this book.

1.4 Organization of the Book

Towards the direction of building the theoretical foundation of EAs, and consequently, designing better evolutionary learning algorithms, this book is composed of four parts, covering the preliminaries, analysis methodology, theoretical perspectives, and learning algorithms.

In Part I, we briefly introduce evolutionary learning and some preliminaries for theoretical studies.

In Part II, aiming at analyzing running time complexity and approximation ability, which are two essential theoretical aspects of randomized search heuristics [Neumann and Witt, 2010, Auger and Doerr, 2011], we present two general approaches, i.e., *convergence-based analysis* and *switch analysis*, for analyzing running time bounds of EAs, and a general framework SEIP for characterizing the approximation performance. These serve as general tools for attaining the theoretical results reported in the remainder of the book.

In Part III, we present a series of theoretical results of EAs. We first study how to identify boundary problems of a *problem class* for a given EA, that is, to find the easiest and hardest problems. Then, we study the influence of major factors of EAs on performance, including recombination, solution representation, inaccurate fitness evaluation and population. Finally, we study EAs for constrained optimization, which is often encountered in real-world learning tasks.

In Part IV, we present a series of evolutionary learning algorithms with bounded approximation guarantees, inspired by the theoretical results derived in Part III. We start from *selective ensemble*, trying to select a subset of individual learners to achieve better ensemble generalization performance. Our Pareto optimization algorithm attempts to optimize the generalization

performance and minimize the number of selected individual learners simultaneously, behaving at least as well as state-of-the-art selective ensemble algorithms. We then study a more general problem, *subset selection*, i.e., to select a limited number of items optimizing some given objective function. Our Pareto optimization algorithm can achieve the best known polynomial-time approximation guarantee. We also present variants of the Pareto optimization algorithm achieving the best known polynomial-time approximation guarantee for two extended subset selection problems. Finally, considering that real-world machine learning tasks are usually not noise-free and the scale can be quite large, we present noise-tolerant and parallel algorithms for subset selection.

We wish that our general theoretical tools presented in Part II can be found useful for interested readers to explore the theoretical foundation of evolutionary learning, the theoretical results presented in Part III can be found helpful for readers to get more understanding about behaviors of evolutionary learning processes and offer some insight for algorithm design, and the algorithms presented in Part IV can be found useful in various real-world learning applications.



Preliminaries

This chapter will first introduce some basic EAs, the optimization engine of evolutionary learning. At the beginning of theoretical studies of EAs, some simple pseudo-Boolean functions were used as the step stones, which are also illustrative for the theoretical understanding of evolutionary learning algorithms. Then this chapter will introduce such functions, followed by some basic knowledge about the running time complexity of algorithms. Finally, this chapter will model EAs as Markov chains, a useful mathematical model, and introduce two classical approaches for running time analysis, i.e., *fitness level* and *drift analysis*.

2.1 Evolutionary Algorithms

EAs [Bäck, 1996] are a type of general-purpose heuristic optimization algorithms, which simulate the natural evolution process by considering two key factors, variational reproduction, and superior selection. They repeatedly reproduce solutions, by varying currently maintained ones, and eliminate inferior solutions such that solutions are improved iteratively. Though many variants exist, EAs share a common procedure, as described in Figure 1.2; that is,

1. Generate an initial population of solutions;
2. Reproduce new solutions based on the current population by mutation and recombination;
3. Evaluate the newly generated offspring solutions;
4. Form a new population by selecting good solutions from the current population and the offspring solutions;
5. Repeat from Step 2 until some stop criterion is met.

In the following, we introduce some specific EAs that have been widely used in the running time analysis of EAs [Neumann and Witt, 2010, Auger and Doerr, 2011].

The (1+1)-EA algorithm reflects the common structure of EAs. As presented in Algorithm 2.1, it is a simple EA for maximizing pseudo-Boolean functions over $\{0, 1\}^n$. It maintains only one solution, i.e., uses a population size of 1, and repeatedly improves the current solution by using bit-wise mutation in line 3 and selection in lines 4-6. Bit-wise mutation flips each bit of a solution independently with probability $p \in (0, 0.5)$. When the bit-wise mutation operator is replaced by the one-bit mutation operator, which flips a randomly chosen bit of a solution, the algorithm is called *randomized local search* (RLS). When the selection step is strict, i.e., line 4 changes to “**if** $f(s') > f(s)$ **then**”, the algorithms are denoted by (1+1)-EA $^\neq$ and RLS $^\neq$, respectively.

Algorithm 2.1 (1+1)-EA Algorithm

Input: pseudo-Boolean function $f : \{0, 1\}^n \rightarrow \mathbb{R}$

Output: solution from $\{0, 1\}^n$

Process:

- 1: let s = a solution uniformly and randomly selected from $\{0, 1\}^n$;
 - 2: **while** criterion is not met **do**
 - 3: apply bit-wise mutation on s to generate s' ;
 - 4: **if** $f(s') \geq f(s)$ **then**
 - 5: $s = s'$
 - 6: **end if**
 - 7: **end while**
 - 8: **return** s
-

The above-mentioned two commonly used mutation operators are summarized as below:

One-bit mutation: flip one randomly selected bit of s .

Bit-wise mutation: flip each bit of s with probability $p \in (0, 0.5)$.

Note that when the mutation probability p is not explicitly indicated, it is assumed to be the common setting $p = 1/n$.

The $(\mu+1)$ -EA algorithm, as presented in Algorithm 2.2, applies a parent population size of μ . It starts from an initial population of μ random solutions in line 1. In each iteration, it generates one new solution s' by mutating a solution randomly selected from the current population in lines 3-4, and then uses s' to replace the worst solution in the population if s' is not worse in lines 5-8.

The $(1+\lambda)$ -EA algorithm, as presented in Algorithm 2.3, applies an offspring population size of λ . It starts from a random solution in line 1. In each iteration, it first generates λ offspring solutions by independently mutating the current solution for λ times in lines 3-5, and then selects the best from the current solution and offspring solutions as the next solution in lines 6-8.

Algorithm 2.2 $(\mu+1)$ -EA Algorithm

Input: pseudo-Boolean function $f : \{0, 1\}^n \rightarrow \mathbb{R}$; positive integer $\mu \in \mathbb{N}^+$ **Output:** μ solutions from $\{0, 1\}^n$ **Process:**

```

1: let  $P = \mu$  solutions uniformly and randomly selected from  $\{0, 1\}^n$ ;
2: while criterion is not met do
3:   select a solution  $s$  from  $P$  uniformly at random;
4:   apply bit-wise mutation on  $s$  to generate  $s'$ ;
5:   let  $z = \arg \min_{z \in P} f(z)$ , where ties are broken randomly;
6:   if  $f(s') \geq f(z)$  then
7:      $P = (P \setminus \{z\}) \cup \{s'\}$ 
8:   end if
9: end while
10: return  $P$ 

```

Algorithm 2.3 $(1+\lambda)$ -EA Algorithm

Input: pseudo-Boolean function $f : \{0, 1\}^n \rightarrow \mathbb{R}$; positive integer $\lambda \in \mathbb{N}^+$ **Output:** solution from $\{0, 1\}^n$ **Process:**

```

1: let  $s$  = a solution uniformly and randomly selected from  $\{0, 1\}^n$ ;
2: while criterion is not met do
3:   for  $i = 1$  to  $\lambda$ 
4:     apply bit-wise mutation on  $s$  to generate  $s_i$ 
5:   end for
6:   if  $\max\{f(s_1), \dots, f(s_\lambda)\} \geq f(s)$  then
7:      $s = \arg \max_{s' \in \{s_1, \dots, s_\lambda\}} f(s')$ 
8:   end if
9: end while
10: return  $s$ 

```

The $(\mu+\lambda)$ -EA algorithm, as presented in Algorithm 2.4, is a general population-based EA with a parent population size of μ and an offspring population size of λ . It maintains μ solutions. In each iteration, one solution selected from the current population is used to generate an offspring solution by bit-wise mutation. This process is repeated independently for λ times, and then μ solutions out of the parent and offspring solutions are selected to form the next population. Note that the selection strategies for reproducing new solutions and updating the population can be arbitrary. Thus, $(\mu+\lambda)$ -EA is quite general, and covers most population-based EAs in previous theoretical analyses, e.g., [He and Yao, 2002, Chen et al., 2012, Lehre and Yao, 2012].

The above introduced EAs are used for single-objective optimization. The SEMO algorithm [Laumanns et al., 2004], as described in Algorithm 2.5, is a simple EA for maximizing multiple pseudo-Boolean functions simultaneously. It is also the first analyzed MOEA due to its simplicity, which ex-

Algorithm 2.4 $(\mu+\lambda)$ -EA Algorithm

Input: pseudo-Boolean function $f : \{0, 1\}^n \rightarrow \mathbb{R}$; two positive integers $\mu \in \mathbb{N}^+$ and $\lambda \in \mathbb{N}^+$

Output: μ solutions from $\{0, 1\}^n$

Process:

- 1: let $P = \mu$ solutions uniformly and randomly selected from $\{0, 1\}^n$;
 - 2: **while** criterion is not met **do**
 - 3: **for** $i = 1$ to λ
 - 4: select a solution s from P according to some selection mechanism;
 - 5: apply bit-wise mutation on s to generate s_i
 - 6: **end for**
 - 7: $P = \mu$ solutions selected from the μ parent solutions and λ offspring solutions $\{s_1, \dots, s_\lambda\}$ according to some strategy
 - 8: **end while**
 - 9: **return** P
-

plains the common structure of various MOEAs. SEMO employs the one-bit mutation operator to generate a new solution by line 5 of Algorithm 2.5, and keeps only non-dominated solutions in the population by lines 6-8.

Algorithm 2.5 SEMO Algorithm

Input: pseudo-Boolean function vector $f = (f_1, f_2, \dots, f_m)$

Output: set of solutions from $\{0, 1\}^n$

Process:

- 1: let s = a solution uniformly and randomly selected from $\{0, 1\}^n$;
 - 2: let $P = \{s\}$;
 - 3: **while** criterion is not met **do**
 - 4: select a solution s from P uniformly at random;
 - 5: apply one-bit mutation on s to generate s' ;
 - 6: **if** $\nexists z \in P$ such that $z \succeq s'$ **then**
 - 7: $P = (P \setminus \{z \in P \mid s' \succ z\}) \cup \{s'\}$
 - 8: **end if**
 - 9: **end while**
 - 10: **return** P
-

There are some famous variants of SEMO. FEMO [Laumanns et al., 2004] accelerates the exploration of the optimal set in SEMO by using a fair sampling strategy, which makes every solution produce the same number of offspring solutions in the end. GEMO [Laumanns et al., 2004] extends FEMO to achieve the maximum progress towards the Pareto front by using a greedy selection mechanism, which allows only the newly included solution dominating some current solutions to have reproduction opportunities. All these MOEAs use the one-bit mutation operator. By replacing the one-bit mutation operator which searches locally in SEMO by the bit-

wise mutation operator which searches globally, the global SEMO (GSEMO) algorithm [Giel, 2003] is obtained; that is, GSEMO is the same as SEMO except for that line 5 in Algorithm 2.5 changes to “apply bit-wise mutation on s to generate s' ”.

2.2 Pseudo-Boolean Functions

The pseudo-Boolean function class in Definition 2.1 is a large function class which only requires the solution space to be $\{0, 1\}^n$ and the objective space to be \mathbb{R} . Many well-known NP-hard problems, e.g., the vertex cover and 0-1 knapsack problems, belong to this class.

Definition 2.1 (Pseudo-Boolean Function). *A pseudo-Boolean function has the form: $f : \{0, 1\}^n \rightarrow \mathbb{R}$.*

The UBoolean function class in Definition 2.2 is a broad class of non-trivial pseudo-Boolean functions, where the global optimal solution to each function is unique. For any function in UBoolean, one can assume without loss of generality that the global optimal solution is $(1, 1, \dots, 1)$, abbreviated as 1^n . This is because EAs treat the bits 0 and 1 symmetrically, and thus the 0-bits in an optimal solution can be interpreted as 1-bits without affecting the behavior of EAs.

Definition 2.2 (UBoolean). *A pseudo-Boolean function $f : \{0, 1\}^n \rightarrow \mathbb{R}$ in UBoolean satisfies*

$$\exists s \in \{0, 1\}^n, \forall s' \in \{0, 1\}^n \setminus \{s\} : f(s') < f(s). \quad (2.1)$$

Various pseudo-Boolean problems with different structures have been used to disclose properties of EAs, e.g., in [Droste et al., 1998, He and Yao, 2001, Droste et al., 2002]. In the following, we introduce some pseudo-Boolean problems that will be used in this book.

The OneMax problem as presented in Definition 2.3 aims at maximizing the number of 1-bits of a solution. Its optimal solution is 1^n with function value n . It has been shown that the expected running time of (1+1)-EA on OneMax is $\Theta(n \log n)$ [Droste et al., 2002], where Θ will be defined at the end of Section 2.3.

Definition 2.3 (OneMax). *The OneMax problem of size n is to find an n bits binary string which maximizes*

$$f(s) = \sum_{i=1}^n s_i, \quad (2.2)$$

where s_i denotes the i -th bit of $s \in \{0, 1\}^n$.

The LeadingOnes problem as presented in Definition 2.4 aims at maximizing the number of consecutive 1-bits counting from the left of a solution. Its optimal solution is 1^n with function value n . For (1+1)-EA solving LeadingOnes, the expected running time is $\Theta(n^2)$ [Droste et al., 2002].

Definition 2.4 (LeadingOnes). *The LeadingOnes problem of size n is to find an n bits binary string which maximizes*

$$f(\mathbf{s}) = \sum_{i=1}^n \prod_{j=1}^i s_j, \quad (2.3)$$

where s_j denotes the j -th bit of $\mathbf{s} \in \{0, 1\}^n$.

The Peak problem as presented in Definition 2.5 has the same fitness for all solutions except for the optimal solution 1^n . It has been shown that for solving this problem, (1+1)-EA requires $2^{\Omega(n)}$ running time with an overwhelming probability [Oliveto and Witt, 2011], where Ω will be defined at the end of Section 2.3.

Definition 2.5 (Peak). *The Peak problem of size n is to find an n bits binary string which maximizes*

$$f(\mathbf{s}) = \prod_{i=1}^n s_i, \quad (2.4)$$

where s_i denotes the i -th bit of $\mathbf{s} \in \{0, 1\}^n$.

The Trap problem as presented in Definition 2.6 aims at maximizing the number of 0-bits of a solution except for the optimal solution 1^n . Its optimal function value is $c - n > 0$, and the function value for any non-optimal solution is no greater than 0. The expected running time of (1+1)-EA on Trap has been proved to be $\Theta(n^n)$ [Droste et al., 2002].

Definition 2.6 (Trap). *The Trap problem of size n is to find an n bits binary string which maximizes*

$$f(\mathbf{s}) = c \cdot \prod_{i=1}^n s_i - \sum_{i=1}^n s_i, \quad (2.5)$$

where $c > n$ and s_i denotes the i -th bit of $\mathbf{s} \in \{0, 1\}^n$.

Next, we introduce two bi-objective pseudo-Boolean problems, Leading Ones Trailing Zeros (LOTZ) and Count Ones Count Zeros (COCZ); both are to maximize two pseudo-Boolean functions simultaneously. They have been widely used to investigate the properties of MOEAs [Giel, 2003, Lammanns et al., 2004], and will be used in this book.

For LOTZ, the first objective is to maximize the number of leading 1-bits, the same as the LeadingOnes problem, and the other objective is to maximize the number of trailing 0-bits. As analyzed in [Laumanns et al., 2004], the objective space of LOTZ can be partitioned into $(n + 1)$ subsets F_0, F_1, \dots, F_n , where $i \in \{0, 1, \dots, n\}$ is the sum of the two objective values, i.e., $\mathbf{f}(\mathbf{s}) \in F_i$ if $f_1(\mathbf{s}) + f_2(\mathbf{s}) = i$. $F_n = \{(0, n), (1, n - 1), \dots, (n, 0)\}$ is the Pareto front, and the corresponding Pareto optimal solutions are $0^n, 1^{n-1}, \dots, 1^n$.

Definition 2.7 (LOTZ). *The LOTZ problem of size n is to find n bits binary strings which maximize*

$$\mathbf{f}(\mathbf{s}) = \left(\sum_{i=1}^n \prod_{j=1}^i s_j, \sum_{i=1}^n \prod_{j=i}^n (1 - s_j) \right), \quad (2.6)$$

where s_j denotes the j -th bit of $\mathbf{s} \in \{0, 1\}^n$.

For COCZ, the first objective is to maximize the number of 1-bits, the same as the OneMax problem, and the other objective is to maximize the number of 1-bits in the first half of the solution plus the number of 0-bits in the second half. The two objectives are consistent in maximizing the number of 1-bits in the first half of the solution, but conflict in the second half. As analyzed in [Laumanns et al., 2004], the objective space of COCZ can be partitioned into $(n/2 + 1)$ subsets $F_0, F_1, \dots, F_{n/2}$, where $i \in \{0, 1, \dots, n/2\}$ is the number of 1-bits in the first half of the solution. Each F_i contains $(n/2 + 1)$ different objective vectors $(i + j, i + n/2 - j)$, where $j \in \{0, 1, \dots, n/2\}$ is the number of 1-bits in the second half. The Pareto front is $F_{n/2} = \{(n/2, n), (n/2 + 1, n - 1), \dots, (n, n/2)\}$, and the set of all Pareto optimal solutions is $\{1^{\frac{n}{2}} * 0^{\frac{n}{2}} \mid * \in \{0, 1\}\}$, with the size of $2^{\frac{n}{2}}$.

Definition 2.8 (COCZ). *The COCZ problem of size n is to find n bits binary strings which maximize*

$$\mathbf{f}(\mathbf{s}) = \left(\sum_{i=1}^n s_i, \sum_{i=1}^{n/2} s_i + \sum_{i=n/2+1}^n (1 - s_i) \right), \quad (2.7)$$

where n is even and s_i denotes the i -th bit of $\mathbf{s} \in \{0, 1\}^n$.

2.3 Running Time Complexity

To measure the performance of an algorithm optimizing a problem, the running time complexity is often employed, and has been a fundamental theoretical aspect for randomized search heuristics [Neumann and Witt,

2010, Auger and Doerr, 2011]. For single-objective optimization, the running time of EAs is usually defined as the number of fitness evaluations, i.e., computing $f(\cdot)$, until a desired solution is found for the first time, because the fitness evaluation is often the computational process with the highest cost of the algorithm [Neumann and Witt, 2010, Auger and Doerr, 2011].

For exact analysis, the desired solutions are optimal solutions. For approximate analysis, the desired solutions are solutions with some approximation guarantee, as defined in Definition 2.9. Here, maximization is considered, while α -approximation can be defined similarly for minimization with $\alpha \geq 1$.

Definition 2.9 (α -Approximation). *Given an objective function f to be maximized, a solution s is said to have an α -approximation ratio if $f(s) \geq \alpha \cdot \text{OPT}$, where $\alpha \in [0, 1]$.*

When EAs are used for noisy optimization, the running time is usually defined as the number of fitness evaluations needed to find the desired solution with respect to the true fitness function f for the first time [Droste, 2004, Akimoto et al., 2015, Gießen and Kötzing, 2016].

For multi-objective optimization, the running time of MOEAs is counted as the number of calls to f until finding the Pareto front or an approximation of the Pareto front. To find the Pareto front means that at least one corresponding solution is found for each objective vector in the Pareto front. Approximation for multi-objective maximization is defined in Definition 2.10. To find an approximation of the Pareto front means that for every Pareto optimal solution, we have a solution that is α -close on every objective function.

Definition 2.10 (α -Approximation for Multi-objective Optimization). *Given m objective functions f_1, f_2, \dots, f_m to be maximized simultaneously, a set P of solutions is said to have an α -approximation ratio if $\forall (q_1^*, q_2^*, \dots, q_m^*) \in F^*$, i.e., for any objective vector in the Pareto front, $\exists s \in P$ such that $\forall i \in [m] : f_i(s) \geq \alpha \cdot q_i^*$, where $\alpha \in [0, 1]$ and F^* denotes the Pareto front.*

As EAs are randomized algorithms, *expected* running time is considered. If without explicit mention of approximate analysis, we are assumed to consider expected running time for exact analysis, i.e., the expected running time until finding an optimal solution for single-objective optimization or finding the Pareto front for multi-objective optimization.

In the following, we introduce notations $O, o, \Omega, \omega, \Theta$, which are used to characterize how the running time of an algorithm grows with the problem size. Let f and g be two functions defined on real numbers.

- $f(n) = O(g(n))$ iff \exists constant $m \in \mathbb{R}^+, n_0 \in \mathbb{R}$ such that $\forall n \geq n_0 : |f(n)| \leq m \cdot |g(n)|$.
- $f(n) = o(g(n))$ iff \forall constant $m \in \mathbb{R}^+, \exists n_0 \in \mathbb{R}$ such that $\forall n \geq n_0 : |f(n)| \leq m \cdot |g(n)|$.

- $f(n) = \Omega(g(n))$ iff $g(n) = O(f(n))$.
- $f(n) = \omega(g(n))$ iff $g(n) = o(f(n))$.
- $f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

2.4 Markov Chain Modeling

During the running of an EA, the offspring solutions are often generated by varying the current solutions. Thus, given the current solutions, the offspring solutions are drawn from a fixed distribution, regardless of how the current solutions are generated. This process can be naturally modeled by a Markov chain, which has been widely used for the analysis of EAs [He and Yao, 2001, 2003a]. A Markov chain is a sequence of random variables, $\{\xi_t\}_{t=0}^{+\infty}$, where the variable ξ_{t+1} depends only on the variable ξ_t , i.e., $P(\xi_{t+1} | \xi_t, \xi_{t-1}, \dots, \xi_0) = P(\xi_{t+1} | \xi_t)$. Thus, a Markov chain can be fully described by its initial state ξ_0 and the transition probability $P(\xi_{t+1} | \xi_t)$.

Denote S as the solution space of a problem. An EA maintaining m solutions (i.e., the population size is m) has a search space $\mathcal{X} \subseteq S^m$, of which the exact size can be found in [Suzuki, 1995]. There are several possible ways to model EAs as Markov chains. The most straightforward way is to take \mathcal{X} as the state space of the Markov chain, and the chain is denoted as $\{\xi_t\}_{t=0}^{+\infty}$ where $\xi_t \in \mathcal{X}$. Let $\mathcal{X}^* \subset \mathcal{X}$ denote the target state space; for example, for exact analysis in single-objective optimization, one population belongs to \mathcal{X}^* iff it contains at least one optimal solution. It is clear that a Markov chain models an EA process, i.e., the process of running an EA on a problem instance. As shown in Figure 2.1, the population P_t after running t iterations of the EA corresponds to the state ξ_t of the Markov chain at time t . Note that discrete state space, i.e., discrete \mathcal{X} , will always be considered.

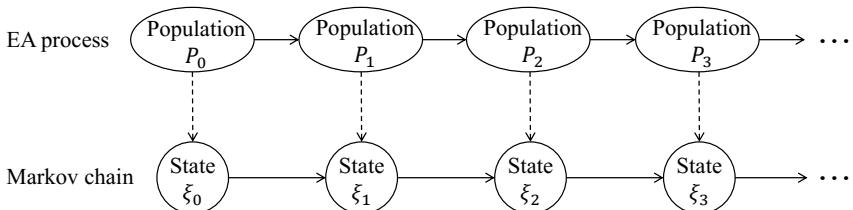


Figure 2.1. Markov chain modeling of an EA process, where P_i denotes the population after running i iterations of the EA.

The goal of the analysis is to disclose how soon the chain $\{\xi_t\}_{t=0}^{+\infty}$ (and thus, the corresponding EA process) falls into \mathcal{X}^* from some initial state. We then define the first hitting time of a Markov chain, which counts the number of steps needed to reach the target state space for the first time.

Note that \mathcal{X} and \mathcal{X}^* will always be used to denote the state space and the target state space, respectively.

Definition 2.11 (Conditional First Hitting Time). *Given a Markov chain $\{\xi_t\}_{t=0}^{+\infty}$, starting from time t_0 where $\xi_{t_0} = x$, let τ be a random variable that denotes the first hitting time:*

$$\tau = 0 : \xi_{t_0} \in \mathcal{X}^*, \quad (2.8)$$

$$\tau = i : \xi_{t_0+i} \in \mathcal{X}^* \wedge \xi_{t_0+j} \notin \mathcal{X}^* \forall j \in \{0, 1, \dots, i-1\}. \quad (2.9)$$

The conditional expectation of τ ,

$$\mathbb{E}[\tau \mid \xi_{t_0} = x] = \sum_{i=0}^{+\infty} i \cdot P(\tau = i \mid \xi_{t_0} = x), \quad (2.10)$$

is called the *conditional first hitting time (CFHT)* of the Markov chain starting from $t = t_0$ and $\xi_{t_0} = x$.

Definition 2.12 (Distribution-CFHT). *Given a Markov chain $\{\xi_t\}_{t=0}^{+\infty}$, starting from time t_0 where ξ_{t_0} is drawn from a state distribution π , the expectation of the CFHT,*

$$\mathbb{E}[\tau \mid \xi_{t_0} \sim \pi] = \mathbb{E}_{x \sim \pi}[\tau \mid \xi_{t_0} = x] = \sum_{x \in \mathcal{X}} \pi(x) \cdot \mathbb{E}[\tau \mid \xi_{t_0} = x], \quad (2.11)$$

is called the *distribution-CFHT (DCFHT)* of the Markov chain starting from $t = t_0$ and $\xi_{t_0} \sim \pi$.

A Markov chain is called an *absorbing* one if it never goes out of the target state once found.

Definition 2.13 (Absorbing Markov Chain). *A Markov chain $\{\xi_t\}_{t=0}^{+\infty}$ is said to be absorbing, if*

$$\forall x \in \mathcal{X}^*, \forall t \geq 0 : P(\xi_{t+1} \neq x \mid \xi_t = x) = 0. \quad (2.12)$$

Given any non-absorbing Markov chain, one can always construct a corresponding absorbing chain which simulates the non-absorbing chain but stays in the target state once it has been found. Thus, the first hitting time of the constructed absorbing chain is the same as that of the corresponding non-absorbing chain. For simplicity of discussion, assume that an EA can always be modeled by an absorbing Markov chain.

If an EA employs time-invariant operators, it can be modeled by a *homogeneous* Markov chain, i.e., a Markov chain whose state transition probabilities do not depend on time t .

Definition 2.14 (Homogeneous Markov Chain). *A Markov chain $\{\xi_t\}_{t=0}^{+\infty}$ is said to be homogeneous, if*

$$\forall t \geq 0, \forall x, y \in \mathcal{X} : P(\xi_{t+1} = y \mid \xi_t = x) = P(\xi_1 = y \mid \xi_0 = x). \quad (2.13)$$

Next, we introduce some properties about the CFHT and DCFHT of Markov chains [Norris, 1997].

Lemma 2.1. *Given a Markov chain $\{\xi_t\}_{t=0}^{+\infty}$, we have*

$$\forall x \in \mathcal{X}^* : \mathbb{E}[\tau \mid \xi_t = x] = 0, \quad (2.14)$$

$$\begin{aligned} \forall x \notin \mathcal{X}^* : \mathbb{E}[\tau \mid \xi_t = x] \\ = 1 + \sum_{y \in \mathcal{X}} P(\xi_{t+1} = y \mid \xi_t = x) \mathbb{E}[\tau \mid \xi_{t+1} = y]. \end{aligned} \quad (2.15)$$

Lemma 2.2. *Given an absorbing Markov chain $\{\xi_t\}_{t=0}^{+\infty}$, we have*

$$\begin{aligned} \mathbb{E}[\tau \mid \xi_t \sim \pi_t] &= \mathbb{E}_{x \sim \pi_t} [\tau \mid \xi_t = x] \\ &= 1 - \pi_t(\mathcal{X}^*) + \sum_{x \in \mathcal{X} \setminus \mathcal{X}^*, y \in \mathcal{X}} \pi_t(x) P(\xi_{t+1} = y \mid \xi_t = x) \mathbb{E}[\tau \mid \xi_{t+1} = y] \\ &= 1 - \pi_t(\mathcal{X}^*) + \mathbb{E}[\tau \mid \xi_{t+1} \sim \pi_{t+1}], \end{aligned} \quad (2.16)$$

where $\pi_{t+1}(x) = \sum_{y \in \mathcal{X}} \pi_t(y) P(\xi_{t+1} = x \mid \xi_t = y)$ is the distribution of ξ_{t+1} .

Lemma 2.3. *Given a homogeneous Markov chain $\{\xi_t\}_{t=0}^{+\infty}$, we have*

$$\forall t_1, t_2 \geq 0, x \in \mathcal{X} : \mathbb{E}[\tau \mid \xi_{t_1} = x] = \mathbb{E}[\tau \mid \xi_{t_2} = x]. \quad (2.17)$$

For an EA solving a problem, the running time can be counted as the number of calls to the fitness evaluation until a target population is met for the first time. For the corresponding Markov chain, the first hitting time corresponds to the number of generations. Within one generation, an EA takes time to manipulate and evaluate solutions that relates to the number of solutions it maintains. Thus, the expected running time of an EA starting from ξ_0 and that starting from $\xi_0 \sim \pi_0$ are respectively equal to

$$m_1 + m_2 \cdot \mathbb{E}[\tau \mid \xi_0] \quad \text{and} \quad m_1 + m_2 \cdot \mathbb{E}[\tau \mid \xi_0 \sim \pi_0], \quad (2.18)$$

where m_1 and m_2 are the number of fitness evaluations for the initial population and each iteration, respectively. For example, for (1+1)-EA, $m_1 = 1$ and $m_2 = 1$; for (1+ λ)-EA, $m_1 = 1$ and $m_2 = \lambda$. Note that if the initial population is not specified, it is assumed by default a uniform initial distribution π_u , i.e., $m_1 + m_2 \cdot \mathbb{E}[\tau \mid \xi_0 \sim \pi_u] = m_1 + m_2 \cdot \sum_{x \in \mathcal{X}} (1/|\mathcal{X}|) \cdot \mathbb{E}[\tau \mid \xi_0 = x]$.

2.5 Analysis Tools

To analyze the running time of an EA solving a problem, one can model the EA as a Markov chain and then analyze its first hitting time. In this section, we introduce some common methods for analyzing the first hitting time of Markov chains.

Fitness level [Wegener, 2002] is an intuitive method for analyzing the expected running time of *elitist* EAs where the best solution won't be lost once found. Given an optimization problem, the solution space is partitioned into level sets according to the fitness values, and the level sets are then ordered according to the fitness of solutions in the sets. This partition is formally described in Definition 2.15 for maximization problems.

Definition 2.15 ($<_f$ -Partition). *Given a maximization problem $f : \mathcal{S} \rightarrow \mathbb{R}$ and the solution space \mathcal{S} with target subspace \mathcal{S}^* , $\forall \mathcal{S}_1, \mathcal{S}_2 \subseteq \mathcal{S}$, the relation $\mathcal{S}_1 <_f \mathcal{S}_2$ holds if $\forall s \in \mathcal{S}_1, s' \in \mathcal{S}_2 : f(s) < f(s')$. A $<_f$ -partition of \mathcal{S} is a partition of \mathcal{S} into non-empty sets $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_m$ such that $\mathcal{S}_1 <_f \mathcal{S}_2 <_f \dots <_f \mathcal{S}_m$ and $\mathcal{S}_m = \mathcal{S}^*$.*

Note that elitist EAs select solutions with better fitness. The level sets, intuitively, form stairs, based on which an upper bound of the running time can be derived by summing up the maximum time taken for leaving every stair, i.e., by pessimistically assuming that a single jump can reach only the adjacent upper-level set, as shown in Figure 2.2(a). Similarly, a lower bound is the minimum time of leaving a stair, i.e., by optimistically assuming that a single jump can reach the target level set, as shown in Figure 2.2(b). This intuition is formally described in Theorem 2.1, with a slightly modified but equivalent form from the original definition to unify the lower and upper bounds. In the theorem, when the EA uses a population, the notation $\xi_t \in \mathcal{S}_i$ conveys the meaning that the best solution in the population ξ_t is in the solution space \mathcal{S}_i .

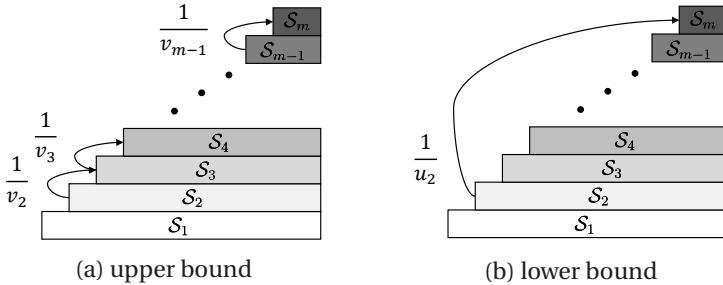


Figure 2.2. Illustration of the fitness level method, where v_i and u_i denote the lower and upper bounds of the probability of leaving the level set \mathcal{S}_i , respectively.

Theorem 2.1 (Fitness Level [Wegener, 2002]). *For an elitist EA solving a problem f modeled by a Markov chain $\{\xi_t\}_{t=0}^{+\infty}$, let $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_m$ be a $<_f$ -partition. $\forall x \in \mathcal{S}_i$, let $v_i \leq P(\xi_{t+1} \in \cup_{j=i+1}^m \mathcal{S}_j \mid \xi_t = x)$ and $u_i \geq P(\xi_{t+1} \in \cup_{j=i+1}^m \mathcal{S}_j \mid \xi_t = x)$. The DCFHT of the chain is at most*

$$\sum_{i=1}^{m-1} \pi_0(\mathcal{S}_i) \cdot \sum_{j=i}^{m-1} \frac{1}{v_j}, \quad (2.19)$$

and is at least

$$\sum_{i=1}^{m-1} \pi_0(\mathcal{S}_i) \cdot \frac{1}{u_i}. \quad (2.20)$$

Theorem 2.2 describes a more elaborated fitness level method.

Theorem 2.2 (Refined Fitness Level [Sudholt, 2013]). *For an elitist EA solving a problem f modeled by a Markov chain $\{\xi_t\}_{t=0}^{+\infty}$, let $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_m$ be a $<_f$ -partition. $\forall x \in \mathcal{S}_i$, let $v_i \leq \min_j \frac{1}{\gamma_{i,j}} P(\xi_{t+1} \in \mathcal{S}_j \mid \xi_t = x)$ and $u_i \geq \max_j \frac{1}{\gamma_{i,j}} P(\xi_{t+1} \in \mathcal{S}_j \mid \xi_t = x)$ where $\sum_{j=i+1}^m \gamma_{i,j} = 1$, and let $\chi_u, \chi_l \in [0, 1]$ be constants such that $\forall i < j < m : \chi_u \geq \gamma_{i,j} / \sum_{k=j}^m \gamma_{i,k} \geq \chi_l$ and $\forall 1 \leq j \leq m - 2 : \chi_u \geq 1 - v_{j+1}/v_j, \chi_l \geq 1 - u_{j+1}/u_j$. The DCFHT of the chain is at most*

$$\sum_{i=1}^{m-1} \pi_0(\mathcal{S}_i) \cdot \left(\frac{1}{v_i} + \chi_u \sum_{j=i+1}^{m-1} \frac{1}{v_j} \right), \quad (2.21)$$

and is at least

$$\sum_{i=1}^{m-1} \pi_0(\mathcal{S}_i) \cdot \left(\frac{1}{u_i} + \chi_l \sum_{j=i+1}^{m-1} \frac{1}{u_j} \right). \quad (2.22)$$

The refined fitness level method follows the general idea of the fitness level method, while introducing a variable χ that reflects the distribution of the probability that the EA jumps to better levels. When χ is small, the EA has a high probability to jump across many levels and thus makes large progress; when χ is large, the EA can take only small progress in every step. χ can take 1 for upper bounds and 0 for lower bounds; this degrades the refined method to the original fitness level method. Thus, the original fitness level method is a special case of the refined one.

Drift analysis is another tool to analyze the DCFHT of Markov chains. It was first introduced to the running time analysis of EAs by He and Yao [2001, 2004]. Since then, it has become a popular tool, and many variants have been developed [Doerr et al., 2012c, Doerr and Goldberg, 2013]. This book will use its *additive* (i.e., Theorem 2.3) as well as *multiplicative* (i.e., Theorem 2.4) version. To use them, a function $V(x)$ has to be constructed to measure the distance of a state x to the target state space \mathcal{X}^* , as illustrated in Figure 2.3. The distance function $V(x)$ satisfies that $V(x \in \mathcal{X}^*) = 0$ and $V(x \notin \mathcal{X}^*) > 0$. The formal definition of a distance function is given in Definition 2.16. Then, the progress on the distance to \mathcal{X}^* in each step,

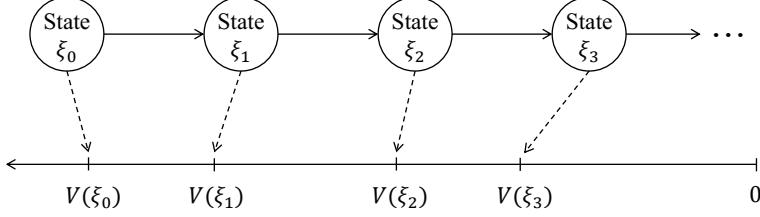


Figure 2.3. Illustration of the drift analysis approach, where $V(\cdot)$ denotes the distance function, i.e., the distance from state \cdot to the target space \mathcal{X}^* .

i.e., $\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t]$, will be considered. For additive drift analysis in Theorem 2.3, an upper (lower) bound of the DCFHT can be derived through dividing the initial distance by a lower (upper) bound of the progress. Multiplicative drift analysis in Theorem 2.4 is much easier to use when the progress is roughly proportional to the current distance to the target space.

Definition 2.16 (Distance Function). For a state space \mathcal{X} with target subspace \mathcal{X}^* , a function V satisfying that $\forall x \in \mathcal{X}^* : V(x) = 0$ and $\forall x \in \mathcal{X} \setminus \mathcal{X}^* : V(x) > 0$ is called a distance function.

Theorem 2.3 (Additive Drift [He and Yao, 2001, 2004]). Given a Markov chain $\{\xi_t\}_{t=0}^{+\infty}$ and a distance function $V(x)$, if $\forall t \geq 0, \xi_t \notin \mathcal{X}^*$, there exists a real number $c_l > 0$ such that

$$\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t] \geq c_l, \quad (2.23)$$

then the DCFHT satisfies

$$\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \leq \sum_{x \in \mathcal{X}} \pi_0(x) \cdot \frac{V(x)}{c_l}; \quad (2.24)$$

if $\forall t \geq 0, \xi_t \notin \mathcal{X}^*$, there exists a real number $c_u > 0$ such that

$$\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t] \leq c_u, \quad (2.25)$$

then the DCFHT satisfies

$$\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \geq \sum_{x \in \mathcal{X}} \pi_0(x) \cdot \frac{V(x)}{c_u}. \quad (2.26)$$

Theorem 2.4 (Multiplicative Drift [Doerr et al., 2012c]). Given a Markov chain $\{\xi_t\}_{t=0}^{+\infty}$ and a distance function $V(x)$, if $\forall t \geq 0, \xi_t \notin \mathcal{X}^*$, there exists a real number $c > 0$ such that

$$\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t] \geq c \cdot V(\xi_t), \quad (2.27)$$

then the DCFHT satisfies

$$\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \leq \sum_{x \in \mathcal{X}} \pi_0(x) \cdot \frac{1 + \log(V(\xi_0)/V_{\min})}{c}, \quad (2.28)$$

where $V_{\min} = \min\{V(x) \mid V(x) > 0\}$.

Theorem 2.5 gives the simplified *negative drift* analysis approach [Oliveto and Witt, 2011, 2012] for proving exponential lower bounds of the first hitting time of Markov chains, where x_t is usually represented by a mapping of ξ_t . It requires two conditions: a constant negative drift, i.e., Eq. (2.29), and exponentially decaying probabilities of jumping towards or away from the target state, i.e., Eq. (2.30). To relax the requirement of a constant negative drift, Theorem 2.6 presents the simplified negative drift analysis approach with self-loops [Rowe and Sudholt, 2014], which takes into account large self-loop probabilities. Theorem 2.7 gives the original negative drift analysis approach [Hajek, 1982], which is stronger because both the two simplified versions can be proved by this original approach.

Theorem 2.5 (Simplified Negative Drift [Oliveto and Witt, 2011, 2012]). Let $x_t, t \geq 0$, be real-valued random variables describing a stochastic process over some state space. Suppose there exists an interval $[a, b] \subseteq \mathbb{R}$, two constants $\delta, \epsilon > 0$ and, possibly depending on $l = b - a$, a function $r(l)$ satisfying $1 \leq r(l) = o(l/\log(l))$ such that $\forall t \geq 0$, the following conditions hold:

$$(1) \quad \mathbb{E}[x_t - x_{t+1} \mid a < x_t < b] \leq -\epsilon, \quad (2.29)$$

$$(2) \quad \forall j \in \mathbb{N}^+ : P(|x_{t+1} - x_t| \geq j \mid x_t > a) \leq \frac{r(l)}{(1 + \delta)^j}. \quad (2.30)$$

Then there is a constant $c > 0$ such that for $T = \min\{t \geq 0 : x_t \leq a \mid x_0 \geq b\}$, it holds that $P(T \leq 2^{cl/r(l)}) = 2^{-\Omega(l/r(l))}$.

Theorem 2.6 (Simplified Negative Drift with Self-loops [Rowe and Sudholt, 2014]). Let $x_t, t \geq 0$, be real-valued random variables describing a stochastic process over some state space. Suppose there exists an interval $[a, b] \subseteq \mathbb{R}$, two constants $\delta, \epsilon > 0$ and, possibly depending on $l = b - a$, a function $r(l)$ satisfying $1 \leq r(l) = o(l/\log(l))$ such that $\forall t \geq 0$, the following conditions hold:

$$(1) \quad \forall a < i < b : \mathbb{E}[x_t - x_{t+1} \mid x_t = i] \leq -\epsilon \cdot P(x_{t+1} \neq i \mid x_t = i), \quad (2.31)$$

$$(2) \quad \forall i > a, j \in \mathbb{N}^+ :$$

$$P(|x_{t+1} - x_t| \geq j \mid x_t = i) \leq \frac{r(l)}{(1 + \delta)^j} \cdot P(x_{t+1} \neq i \mid x_t = i). \quad (2.32)$$

Then there is a constant $c > 0$ such that for $T = \min\{t \geq 0 : x_t \leq a \mid x_0 \geq b\}$, it holds that $P(T \leq 2^{cl/r(l)}) = 2^{-\Omega(l/r(l))}$.

Theorem 2.7 (Negative Drift [Hajek, 1982]). Let x_t , $t \geq 0$, be real-valued random variables describing a stochastic process over some state space. Pick two real numbers $a(l)$ and $b(l)$ depending on a parameter l such that $a(l) < b(l)$. Let $T(l)$ be the random variable denoting the earliest point in time $t \geq 0$ such that $x_t \leq a(l)$. If there are $\lambda(l) > 0$ and $p(l) > 0$ such that

$$\forall t \geq 0 : \mathbb{E} \left[e^{-\lambda(l) \cdot (x_{t+1} - x_t)} \mid a(l) < x_t < b(l) \right] \leq 1 - \frac{1}{p(l)}, \quad (2.33)$$

then for all time bounds $L(l) \geq 0$,

$$\mathbb{P}(T(l) \leq L(l) \mid x_0 \geq b(l)) \leq e^{-\lambda(l) \cdot (b(l) - a(l))} \cdot L(l) \cdot D(l) \cdot p(l), \quad (2.34)$$

where $D(l) = \max \{1, \mathbb{E}[e^{-\lambda(l) \cdot (x_{t+1} - b(l))} \mid x_t \geq b(l)]\}$.

Part II

ANALYSIS METHODOLOGY



Running Time Analysis: Convergence-based Analysis

There has been a significant rise of theoretical studies on EAs during the past two decades, particularly on the running time, which represents the average computational complexity of EAs and is thus a core theoretical issue. Probe problems, e.g., pseudo-Boolean linear problems [Droste et al., 1998], are widely employed to facilitate the analysis on questions such as how efficient EAs can be and what parameters should be used. Interestingly, conflicting conclusions have been reported. For example, using recombination operators in EAs has been shown quite necessary in some problems, e.g., [Jansen and De Jong, 2002, Doerr et al., 2013a, Sudholt, 2017], but is undesired in some other problems, e.g., [Richter et al., 2008]; using a large population has been found helpful in some cases, e.g., [He and Yao, 2002], but helpless in some other cases, e.g., [Chen et al., 2012]. These conflicting results imply the sophisticated situation facing with EAs. When the large variety of problems is considered, it is demanding to have general analysis tools that can guide the analysis of EAs on more problems rather than ad hoc analysis starting from scratch.

We have introduced the fitness level method and the drift analysis approach for analyzing the running time of EAs in Section 2.5. However, we are still short of general tools for theoretically understanding EAs. This chapter presents the *convergence-based analysis* approach [Yu and Zhou, 2008a] for running time analysis of EAs, which can provide a different way for analysis.

The *convergence rate* is an important theoretical issue of EAs, implying how close the current state is to the target state at each step. The convergence issue has been studied for years [Rudolph, 1994, Suzuki, 1995, Rudolph, 1997, He and Kang, 1999, He and Yu, 2001]. In particular, He and Yu [2001] did a thorough study based on the *minorization condition* method [Rosenthal, 1995]. The convergence-based analysis approach is based on a bridge between the expected running time and the convergence rate. As shown in Figure 3.1, they are the expectation and the tail distribution of the first hitting time, respectively. We demonstrate the use of convergence-based analysis by deriving lower bounds of the running time

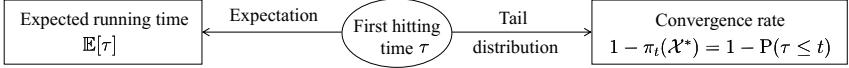


Figure 3.1. Intuition of the convergence-based analysis approach, which builds a bridge between the expected running time and the convergence rate. Note that $\pi_t(\mathcal{X}^*)$ denotes the probability of $\xi_t \in \mathcal{X}^*$, i.e., the probability of the target space \mathcal{X}^* to be reached before time t .

of (1+1)-EA, i.e., Algorithm 2.1, and RLS, i.e., Algorithm 2.1 with one-bit mutation, for solving a hard problem, constrained Trap.

The rest of this chapter is organized as follows. Sections 3.1 and 3.2 present the convergence-based analysis approach and its application illustration, respectively. Section 3.3 concludes this chapter.

3.1 Convergence-based Analysis: Framework

As introduced in Section 2.4, EAs are modeled by Markov chains. A Markov chain $\{\xi_t\}_{t=0}^{+\infty}$ with state space \mathcal{X} is described as “ $\xi \in \mathcal{X}$ ” for simplicity. Let π_t denote the distribution of ξ_t , i.e., the state distribution of the chain at time t .

Definition 3.1 (Convergence). *A Markov chain $\xi \in \mathcal{X}$ with target subspace \mathcal{X}^* is said to converge to \mathcal{X}^* if*

$$\lim_{t \rightarrow +\infty} \pi_t(\mathcal{X}^*) = 1. \quad (3.1)$$

In [He and Yu, 2001], the convergence rate is measured by $(1 - \pi_t(\mathcal{X}^*))$ at time t , which is equivalent to that in [Suzuki, 1995]. Therefore, $1 - \pi_t(\mathcal{X}^*)$ is also used as the measure of convergence rate here.

Definition 3.2 (Convergence Rate). *Given a Markov chain $\xi \in \mathcal{X}$ with target subspace \mathcal{X}^* , the convergence rate to \mathcal{X}^* at time t is $1 - \pi_t(\mathcal{X}^*)$.*

The convergence rate has been studied for many years [Suzuki, 1995, He and Kang, 1999] and recently a general bound has been developed in [He and Yu, 2001] through the minorization condition method [Rosenthal, 1995]. The convergence rate can be bounded by using the *normalized success probability* as in Lemma 3.1, where α_t and β_t are the lower and upper bounds of the normalized success probability, respectively.

Lemma 3.1. *Given an absorbing Markov chain $\xi \in \mathcal{X}$ with target subspace \mathcal{X}^* , if two sequences $\{\alpha_t\}_{t=0}^{+\infty}$ and $\{\beta_t\}_{t=0}^{+\infty}$ satisfy that*

$$\prod_{t=0}^{+\infty} (1 - \alpha_t) = 0, \quad (3.2)$$

$$\beta_t \geq \sum_{x \notin \mathcal{X}^*} P(\xi_{t+1} \in \mathcal{X}^* \mid \xi_t = x) \frac{\pi_t(x)}{1 - \pi_t(\mathcal{X}^*)} \geq \alpha_t, \quad (3.3)$$

then the chain converges to \mathcal{X}^* , and the convergence rate is bounded by †

$$1 - \pi_t(\mathcal{X}^*) \geq (1 - \pi_0(\mathcal{X}^*)) \prod_{i=0}^{t-1} (1 - \beta_i), \quad (3.4)$$

$$1 - \pi_t(\mathcal{X}^*) \leq (1 - \pi_0(\mathcal{X}^*)) \prod_{i=0}^{t-1} (1 - \alpha_i). \quad (3.5)$$

Proof. According to $\pi_t(\mathcal{X}^*) = \sum_{x \in \mathcal{X}^*} \pi_t(x)$ and the absorbing property of the Markov chain, we have

$$\pi_t(\mathcal{X}^*) - \pi_{t-1}(\mathcal{X}^*) = \sum_{x \notin \mathcal{X}^*} \pi_{t-1}(x) P(\xi_t \in \mathcal{X}^* \mid \xi_{t-1} = x). \quad (3.6)$$

By applying Eq. (3.3) to Eq. (3.6), we have

$$(1 - \pi_{t-1}(\mathcal{X}^*)) \alpha_{t-1} \leq \pi_t(\mathcal{X}^*) - \pi_{t-1}(\mathcal{X}^*) \leq (1 - \pi_{t-1}(\mathcal{X}^*)) \beta_{t-1}, \quad (3.7)$$

which implies

$$(1 - \pi_{t-1}(\mathcal{X}^*)) (1 - \alpha_{t-1}) \geq 1 - \pi_t(\mathcal{X}^*) \geq (1 - \pi_{t-1}(\mathcal{X}^*)) (1 - \beta_{t-1}). \quad (3.8)$$

By applying Eq. (3.8) recursively, we have

$$(1 - \pi_0(\mathcal{X}^*)) \prod_{i=0}^{t-1} (1 - \alpha_i) \geq 1 - \pi_t(\mathcal{X}^*) \geq (1 - \pi_0(\mathcal{X}^*)) \prod_{i=0}^{t-1} (1 - \beta_i). \quad (3.9)$$

□

Lemma 3.1 is a variant of the result in [He and Yu, 2001] by using a discrete space. It implies that as far as the probability of an EA “jumping” into the set of target solutions can be estimated for each step, the bounds of its convergence rate can be derived. According to Definition 2.11, the CFHT of a Markov chain is the mathematical expectation of the random variable τ , i.e., the first hitting time. Meanwhile, the cumulative distribution of τ is the probability of a target solution being found before time t ($t = 0, 1, \dots$). Furthermore, we have

$$\pi_{t+1}(\mathcal{X}^*) - \pi_t(\mathcal{X}^*) = \sum_{x \in \mathcal{X}^*} \pi_{t+1}(x) - \sum_{x \in \mathcal{X}^*} \pi_t(x) = P(\tau = t + 1), \quad (3.10)$$

$^\dagger \prod_{i=a}^b (\cdot) = 1$ if $b < a$.

implying that the cumulative distribution of τ is equal to $\pi_t(\mathcal{X}^*)$, which is $(1 - \text{convergence rate})$. In other words, the convergence rate is the complementary cumulative distribution, i.e., tail distribution, of τ . Thus, the convergence rate and the CFHT can be viewed as two sides of a coin, as shown in Figure 3.1.

Meanwhile, the bounds of the cumulative distribution and the expectation of the same random variable have a relationship shown in Lemma 3.2.

Lemma 3.2. *Let u and v denote two discrete random variables that are non-negative integers with limited expectation. $\mathcal{D}_u(\cdot)$ and $\mathcal{D}_v(\cdot)$ denote their cumulative distribution functions, respectively, i.e.,*

$$\mathcal{D}_u(t) = P(u \leq t) = \sum_{i=0}^t P(u = i), \quad (3.11)$$

$$\mathcal{D}_v(t) = P(v \leq t) = \sum_{i=0}^t P(v = i). \quad (3.12)$$

If $\forall t \in \mathbb{N}^{0+}$, $\mathcal{D}_u(t) \geq \mathcal{D}_v(t)$, the expectations of the random variables satisfy

$$\mathbb{E}[u] \leq \mathbb{E}[v], \quad (3.13)$$

where $\mathbb{E}[u] = \sum_{t \in \mathbb{N}^{0+}} t \cdot P(u = t)$ and $\mathbb{E}[v] = \sum_{t \in \mathbb{N}^{0+}} t \cdot P(v = t)$.

Proof. Because \mathcal{D}_u is the cumulative distribution of u , we have

$$\begin{aligned} \mathbb{E}[u] &= 0 \cdot \mathcal{D}_u(0) + \sum_{t=1}^{+\infty} t (\mathcal{D}_u(t) - \mathcal{D}_u(t-1)) = \sum_{i=1}^{+\infty} \sum_{t=i}^{+\infty} (\mathcal{D}_u(t) - \mathcal{D}_u(t-1)) \\ &= \sum_{i=0}^{+\infty} \left(\lim_{t \rightarrow +\infty} \mathcal{D}_u(t) - \mathcal{D}_u(i) \right) = \sum_{i=0}^{+\infty} (1 - \mathcal{D}_u(i)), \end{aligned} \quad (3.14)$$

and similar results can be obtained for v . Thus,

$$\begin{aligned} \mathbb{E}[u] - \mathbb{E}[v] &= \sum_{i=0}^{+\infty} (1 - \mathcal{D}_u(i)) - \sum_{i=0}^{+\infty} (1 - \mathcal{D}_v(i)) \\ &= \sum_{i=0}^{+\infty} (\mathcal{D}_v(i) - \mathcal{D}_u(i)) \leq 0. \end{aligned} \quad (3.15)$$

□

Because $(1 - \text{convergence rate})$ is the cumulative distribution of τ , and the CFHT is the expectation of τ , Lemma 3.2 reveals that the lower/upper bounds of the CFHT can be derived from the upper/lower bounds of the cumulative distribution, and thus from the lower/upper bounds of the convergence rate. Therefore, based on Lemmas 3.1 and 3.2, a pair of general bounds of the CFHT can be obtained, as shown in Theorem 3.1.

Theorem 3.1 (Convergence-based Analysis). *Given an absorbing Markov chain $\xi \in \mathcal{X}$ with target subspace \mathcal{X}^* , let τ denote the first hitting time of ξ , and let π_t denote the distribution of ξ_t . If two sequences $\{\alpha_t\}_{t=0}^{+\infty}$ and $\{\beta_t\}_{t=0}^{+\infty}$ satisfy that*

$$\prod_{t=0}^{+\infty} (1 - \alpha_t) = 0, \quad (3.16)$$

$$\beta_t \geq \sum_{x \notin \mathcal{X}^*} P(\xi_{t+1} \in \mathcal{X}^* \mid \xi_t = x) \frac{\pi_t(x)}{1 - \pi_t(\mathcal{X}^*)} \geq \alpha_t, \quad (3.17)$$

then the chain converges, and the DCFHT is bounded by

$$\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \leq (1 - \pi_0(\mathcal{X}^*)) \left(\sum_{t=1}^{+\infty} t \alpha_{t-1} \prod_{i=0}^{t-2} (1 - \alpha_i) \right), \quad (3.18)$$

$$\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \geq (1 - \pi_0(\mathcal{X}^*)) \left(\sum_{t=1}^{+\infty} t \beta_{t-1} \prod_{i=0}^{t-2} (1 - \beta_i) \right). \quad (3.19)$$

Proof. Applying Lemma 3.1, we have

$$1 - \pi_t(\mathcal{X}^*) \leq (1 - \pi_0(\mathcal{X}^*)) \prod_{i=0}^{t-1} (1 - \alpha_i). \quad (3.20)$$

Considering that $\pi_t(\mathcal{X}^*)$ expresses the cumulative distribution of τ , i.e., $\pi_t(\mathcal{X}^*) = \mathcal{D}_\tau(t)$, the lower bound of $\mathcal{D}_\tau(t)$ can be obtained as

$$\forall t \in \mathbb{N}^{0+} : \mathcal{D}_\tau(t) \geq 1 - (1 - \pi_0(\mathcal{X}^*)) \prod_{i=0}^{t-1} (1 - \alpha_i). \quad (3.21)$$

Imagine a virtual random variable η whose cumulative distribution equals the lower bound of \mathcal{D}_τ . The expectation of η is

$$\begin{aligned} \mathbb{E}[\eta] &= \sum_{t=1}^{+\infty} t \cdot \left((1 - \pi_0(\mathcal{X}^*)) \prod_{i=0}^{t-2} (1 - \alpha_i) - (1 - \pi_0(\mathcal{X}^*)) \prod_{i=0}^{t-1} (1 - \alpha_i) \right) \\ &= \left(\sum_{t=1}^{+\infty} t \alpha_{t-1} \prod_{i=0}^{t-2} (1 - \alpha_i) \right) (1 - \pi_0(\mathcal{X}^*)). \end{aligned} \quad (3.22)$$

Because $\mathcal{D}_\tau(t) \geq \mathcal{D}_\eta(t)$, according to Lemma 3.2, $\mathbb{E}[\tau] \leq \mathbb{E}[\eta]$. Note that $\mathbb{E}[\tau]$ here just corresponds to the DCFHT $\mathbb{E}[\tau \mid \xi_0 \sim \pi_0]$ in Definition 2.12. Thus, the upper bound of the DCFHT is

$$\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \leq \left(\sum_{t=1}^{+\infty} t \alpha_{t-1} \prod_{i=0}^{t-2} (1 - \alpha_i) \right) (1 - \pi_0(\mathcal{X}^*)). \quad (3.23)$$

The lower bound of the DCFHT can be derived similarly. \square

The bounds of the DCFHT, i.e., Eqs. (3.18) and (3.19), have an intuitive explanation. The part $\alpha_{t-1} \prod_{i=0}^{t-2} (1 - \alpha_i)$ (or replacing α by β) indicates the probability of the event that the EA finds a target solution at the t -th step, but does not find it at any earlier step. Theorem 3.1 shows that we can have bounds of the DCFHT from the bounds of the formula

$$\sum_{x \notin \mathcal{X}^*} P(\xi_{t+1} \in \mathcal{X}^* \mid \xi_t = x) \frac{\pi_t(x)}{1 - \pi_t(\mathcal{X}^*)}. \quad (3.24)$$

The first part of Eq. (3.24), i.e., $P(\xi_{t+1} \in \mathcal{X}^* \mid \xi_t = x)$, is the probability of the EA “jumping” into a target population, called *success probability*. The second part, i.e., $\pi_t(x)/(1 - \pi_t(\mathcal{X}^*))$, is a *normalized distribution* over non-target states. As long as these two parts can be estimated, the bounds of the DCFHT can be derived. The more accurate the estimated probabilities, the tighter the derived bounds.

3.2 Convergence-based Analysis: Application Illustration

In this section, we apply the convergence-based analysis approach to derive lower bounds of the expected running time of (1+1)-EA and RLS for solving the constrained Trap problem. The constrained Trap problem is defined below. It has one optimal solution $s^* = (0, 0, \dots, 0, 1)$, abbreviated as $0^{n-1}1$. A solution is called *feasible* if it satisfies the constraint; otherwise it is an *infeasible* solution.

Definition 3.3 (Constrained Trap). Given n weights $w_1 = w_2 = \dots = w_{n-1} > 1$, $w_n = \sum_{i=1}^{n-1} w_i + 1$, and a capacity $c = w_n$, to find s^* such that

$$\begin{aligned} s^* &= \arg \max_{s \in \{0,1\}^n} \sum_{i=1}^n w_i s_i \\ \text{s.t. } &\sum_{i=1}^n w_i s_i \leq c, \end{aligned} \quad (3.25)$$

where s_i denotes the i -th bit of $s \in \{0, 1\}^n$.

The fitness of a solution s is defined as

$$Fitness(s) = \sum_{i=1}^n w_i s_i, \quad (3.26)$$

when s is a feasible solution, i.e., $\sum_{i=1}^n w_i s_i \leq c$, and $Fitness(s) = -c$ otherwise. The fitness function is to be maximized, and the larger the fitness, the better the solution. Here, the maximum fitness value is c .

We consider (1+1)-EA and RLS. As introduced in Section 2.1, the difference between these two algorithms lies in the mutation operator: (1+1)-EA

employs bit-wise mutation, whereas RLS employs one-bit mutation. Because a state x of the corresponding Markov chain corresponds to a solution s , the state space is equal to the solution space, i.e., $\mathcal{X} = \mathcal{S} = \{0, 1\}^n$.

To analyze the bit-wise mutation case in Theorem 3.2, we need to find an upper bound of Eq. (3.24) and then apply Theorem 3.1. We first examine the part of success probability of Eq. (3.24), i.e., $P(\xi_{t+1} \in \mathcal{X}^* | \xi_t = x)$. Assuming a solution has k bits different from the optimal solution, the probability of the solution being mutated to the optimal solution is $p^k(1-p)^{n-k}$. Thus, the maximum probability of a solution being mutated to the optimal solution is $p(1-p)^{n-1}$, implying that there is only one bit difference. Therefore, we have $P(\xi_{t+1} \in \mathcal{X}^* | \xi_t = x) \leq p(1-p)^{n-1}$. Then, by applying Theorem 3.1 with this upper bound, we get the following theorem.

Theorem 3.2. *For (1+1)-EA solving the constrained Trap problem, the expected running time is lower bounded by $\Omega(1/(p(1-p)^{n-1}))$, where $p \in (0, 0.5)$ is the mutation probability.*

Proof. Because $P(\xi_{t+1} \in \mathcal{X}^* | \xi_t = x) \leq p(1-p)^{n-1}$, we have

$$\begin{aligned} \sum_{x \notin \mathcal{X}^*} P(\xi_{t+1} \in \mathcal{X}^* | \xi_t = x) \frac{\pi_t(x)}{1 - \pi_t(\mathcal{X}^*)} &\leq \sum_{x \notin \mathcal{X}^*} p(1-p)^{n-1} \frac{\pi_t(x)}{1 - \pi_t(\mathcal{X}^*)} \\ &= p(1-p)^{n-1}. \end{aligned} \quad (3.27)$$

Let $\beta_t = p(1-p)^{n-1}$. By Theorem 3.1, we have

$$\begin{aligned} \mathbb{E}[\tau | \xi_0 \sim \pi_0] &\geq (1 - \pi_0(\mathcal{X}^*)) \left(\sum_{t=1}^{+\infty} t \beta_{t-1} \prod_{i=0}^{t-2} (1 - \beta_i) \right) \\ &= \left(1 - \frac{1}{2^n}\right) \frac{1}{p} \left(\frac{1}{1-p}\right)^{n-1} = \Omega\left(\frac{1}{p(1-p)^{n-1}}\right), \end{aligned} \quad (3.28)$$

where Eq. (3.28) holds by $\pi_0(\mathcal{X}^*) = 1/2^n$ due to the uniform initial distribution and using algebraic calculation with $\beta_t = p(1-p)^{n-1}$. \square

For the one-bit mutation case, we apply Theorem 3.1 to derive a lower bound $\Omega(2^n)$ shown in Theorem 3.3. There is a trick for calculating Eq. (3.24), which consists of two parts, i.e., success probability and normalized distribution. We divide the state space into subspaces that contain states sharing some common properties, and treat each subspace as a whole. We first divide the state space into $(n + 1)$ subspaces $\{\mathcal{X}_i\}_{i=0}^n$, where \mathcal{X}_i contains all solutions that have exactly i identical bits with the optimal solution, such that solutions in each subspace have the same probability of being mutated to the optimal solution. Through this division, the success probability can be calculated. We then divide the state space into the optimal space \mathcal{X}^* , the feasible space \mathcal{X}_F and the infeasible space \mathcal{X}_I , according to whether solutions satisfy the constraint, and combine this division with the previous one. Through this division, the normalized distribution can be calculated.

Theorem 3.3. For RLS solving the constrained Trap problem, the expected running time is lower bounded by $\Omega(2^n)$.

Proof. Note that a state x is just a solution here. Let

$$\mathcal{X}_i = \{x \in \mathcal{X} \mid \|x - x^*\|_H = n - i\}, \quad (3.29)$$

where $\|\cdot\|_H$ is the Hamming distance and x^* is the optimal solution, implying that solutions in \mathcal{X}_i have i bits identical with the optimal solution, $\mathcal{X} = \cup_{i=0}^n \mathcal{X}_i$, $|\mathcal{X}_i| = \binom{n}{i}$, and $\mathcal{X}_n = \mathcal{X}^*$. According to the one-bit mutation operator, we calculate the success probability

$$\forall x \in \mathcal{X}_i : P(\xi_{t+1} \in \mathcal{X}^* \mid \xi_t = x) = \begin{cases} \frac{1}{n} & \text{if } i = n - 1; \\ 0 & \text{otherwise.} \end{cases} \quad (3.30)$$

Applying Eq. (3.30) to Eq. (3.24), we have

$$\begin{aligned} & \sum_{x \notin \mathcal{X}^*} P(\xi_{t+1} \in \mathcal{X}^* \mid \xi_t = x) \frac{\pi_t(x)}{1 - \pi_t(\mathcal{X}^*)} \\ &= \sum_{x \in \mathcal{X}_{n-1}} P(\xi_{t+1} \in \mathcal{X}^* \mid \xi_t = x) \frac{\pi_t(x)}{1 - \pi_t(\mathcal{X}^*)} \\ &= \frac{1}{n} \sum_{x \in \mathcal{X}_{n-1}} \frac{\pi_t(x)}{1 - \pi_t(\mathcal{X}^*)} = \frac{1}{n} \frac{\pi_t(\mathcal{X}_{n-1})}{1 - \pi_t(\mathcal{X}^*)}. \end{aligned} \quad (3.31)$$

To derive an upper bound of Eq. (3.31), we derive an upper bound of $\pi_t(\mathcal{X}_{n-1})$ and a lower bound of $1 - \pi_t(\mathcal{X}^*)$, respectively.

Let $\mathcal{X} = \mathcal{X}^* \cup \mathcal{X}_F \cup \mathcal{X}_I$, where \mathcal{X}^* contains the optimal solution, i.e., $0^{n-1}1$, \mathcal{X}_F contains all non-optimal feasible solutions whose last bit is 0, and \mathcal{X}_I contains all infeasible solutions whose last bit is 1. Denote

$$\forall i \in \{0, 1, \dots, n-1\} : \mathcal{X}_i^F = \mathcal{X}_i \cap \mathcal{X}_F, \quad (3.32)$$

$$\forall i \in \{1, 2, \dots, n-1\} : \mathcal{X}_i^I = \mathcal{X}_i \cap \mathcal{X}_I. \quad (3.33)$$

Thus, $\mathcal{X}_F = \cup_{i=0}^{n-1} \mathcal{X}_i^F$ and $\mathcal{X}_I = \cup_{i=1}^{n-1} \mathcal{X}_i^I$. By the fitness function, we have

$$\begin{aligned} & \forall x_0 \in \mathcal{X}_0^F, x_1 \in \mathcal{X}_1^F, \dots, x_{n-1} \in \mathcal{X}_{n-1}^F, x_I \in \mathcal{X}_I : \\ & f(x^*) > f(x_0) > f(x_1) > \dots > f(x_{n-1}) > f(x_I). \end{aligned} \quad (3.34)$$

For $\pi_t(\mathcal{X}_{n-1})$, we have

$$\pi_t(\mathcal{X}_{n-1}) = \pi_t(\mathcal{X}_{n-1}^F) + \pi_t(\mathcal{X}_{n-1}^I). \quad (3.35)$$

It is clear that $\mathcal{X}_{n-1}^F = \{0^n\}$. Due to the selection behavior of RLS, i.e., the solution with a worse fitness will be rejected, we have

$$P(\xi_{t+1} \in \mathcal{X}_{n-1}^F \mid \xi_t \in (\mathcal{X}_F \setminus \mathcal{X}_{n-1}^F) \cup \mathcal{X}^*) = 0. \quad (3.36)$$

Given $\xi_t \in \mathcal{X}_{n-1}^F$, i.e., $\xi_t = 0^n$, one bit of ξ_t will be flipped by one-bit mutation and the newly generated offspring solution will always be accepted. Thus, we have

$$P(\xi_{t+1} \in \mathcal{X}_{n-1}^F \mid \xi_t \in \mathcal{X}_{n-1}^F) = 0. \quad (3.37)$$

Because any infeasible solution in \mathcal{X}_I contains at least two 1-bits and one-bit mutation can flip only one bit, the solution 0^n cannot be generated by one step. Thus, we have

$$P(\xi_{t+1} \in \mathcal{X}_{n-1}^F \mid \xi_t \in \mathcal{X}_I) = 0. \quad (3.38)$$

Combining Eqs. (3.36) to (3.38), we have

$$\forall t > 0 : \pi_t(\mathcal{X}_{n-1}^F) = 0. \quad (3.39)$$

Next, we examine $\pi_t(\mathcal{X}_i^I)$. As infeasible solutions have the worst fitness, an infeasible offspring solution can be accepted only if the parent solution is infeasible. Considering the behavior of one-bit mutation, we have

$$\pi_{t+1}(\mathcal{X}_i^I) = \begin{cases} \pi_t(\mathcal{X}_{i+1}^I) \cdot \frac{i}{n} & \text{if } i = 1; \\ \pi_t(\mathcal{X}_{i+1}^I) \cdot \frac{i}{n} + \pi_t(\mathcal{X}_{i-1}^I) \cdot \frac{n-i+1}{n} & \text{if } 2 \leq i \leq n-2; \\ \pi_t(\mathcal{X}_{i-1}^I) \cdot \frac{n-i+1}{n} & \text{if } i = n-1. \end{cases} \quad (3.40)$$

Due to the uniform initial distribution, it holds that $\forall i \in [n-1] : \pi_0(\mathcal{X}_i^I) = \binom{n-1}{i-1} / 2^n$. By Eq. (3.40), we can calculate $\pi_1(\mathcal{X}_i^I)$. For $i = 1$, we have

$$\pi_1(\mathcal{X}_1^I) = \pi_0(\mathcal{X}_2^I) \cdot \frac{1}{n} = \binom{n-1}{1} \frac{1}{2^n} \cdot \frac{1}{n} = \left(1 - \frac{1}{n}\right) \cdot \pi_0(\mathcal{X}_1^I). \quad (3.41)$$

For $2 \leq i \leq n-2$, we have

$$\begin{aligned} \pi_1(\mathcal{X}_i^I) &= \pi_0(\mathcal{X}_{i+1}^I) \cdot \frac{i}{n} + \pi_0(\mathcal{X}_{i-1}^I) \cdot \frac{n-i+1}{n} \\ &= \binom{n-1}{i} \frac{1}{2^n} \cdot \frac{i}{n} + \binom{n-1}{i-2} \frac{1}{2^n} \cdot \frac{n-i+1}{n} \\ &= \left(1 - \frac{1}{n}\right) \cdot \frac{1}{2^n} \cdot \left(\binom{n-2}{i-1} + \binom{n-2}{i-2}\right) \\ &= \left(1 - \frac{1}{n}\right) \cdot \frac{1}{2^n} \cdot \binom{n-1}{i-1} = \left(1 - \frac{1}{n}\right) \cdot \pi_0(\mathcal{X}_i^I). \end{aligned} \quad (3.42)$$

For $i = n-1$, we have

$$\pi_1(\mathcal{X}_{n-1}^I) = \pi_0(\mathcal{X}_{n-2}^I) \cdot \frac{2}{n} = \binom{n-1}{n-3} \frac{1}{2^n} \cdot \frac{2}{n} = \left(1 - \frac{2}{n}\right) \cdot \pi_0(\mathcal{X}_{n-1}^I). \quad (3.43)$$

Combining Eqs. (3.41) to (3.43), we have

$$\forall i \in [n-1] : \pi_1(\mathcal{X}_i^I) \leq \left(1 - \frac{1}{n}\right) \cdot \pi_0(\mathcal{X}_i^I). \quad (3.44)$$

Applying Eq. (3.44) to Eq. (3.40) and repeating the calculation process of Eqs. (3.41) to (3.43), we have

$$\forall t \geq 0, i \in [n-1] : \pi_t(\mathcal{X}_i^I) \leq \left(1 - \frac{1}{n}\right)^t \cdot \pi_0(\mathcal{X}_i^I). \quad (3.45)$$

Applying Eqs. (3.39) and (3.45) to Eq. (3.35), we have

$$\forall t \geq 0 : \pi_t(\mathcal{X}_{n-1}) \leq \pi_0(\mathcal{X}_{n-1}^F) + \pi_0(\mathcal{X}_{n-1}^I) = \frac{n}{2^n}. \quad (3.46)$$

For any solution in $\cup_{j=0}^{n-2} \mathcal{X}_j^F$, because the Hamming distance from the optimal solution is at least 2 and one-bit mutation can flip only one bit, the optimal solution cannot be generated by one step, i.e.,

$$P(\xi_{t+1} \in \mathcal{X}^* \mid \xi_t \in \cup_{j=0}^{n-2} \mathcal{X}_j^F) = 0. \quad (3.47)$$

According to Eq. (3.34) and the selection behavior of RLS, we have

$$P(\xi_{t+1} \in \mathcal{X}_I \cup \mathcal{X}_{n-1}^F \mid \xi_t \in \cup_{j=0}^{n-2} \mathcal{X}_j^F) = 0. \quad (3.48)$$

Combining Eqs. (3.47) and (3.48) leads to

$$P(\xi_{t+1} \in \cup_{j=0}^{n-2} \mathcal{X}_j^F \mid \xi_t \in \cup_{j=0}^{n-2} \mathcal{X}_j^F) = 1. \quad (3.49)$$

Thus, we have

$$\forall t \geq 0 : 1 - \pi_t(\mathcal{X}^*) \geq \pi_t(\cup_{j=0}^{n-2} \mathcal{X}_j^F) \geq \pi_0(\cup_{j=0}^{n-2} \mathcal{X}_j^F) = \frac{2^{n-1} - 1}{2^n}. \quad (3.50)$$

Applying Eqs. (3.46) and (3.50) to Eq. (3.31), we have

$$\sum_{x \notin \mathcal{X}^*} P(\xi_{t+1} \in \mathcal{X}^* \mid \xi_t = x) \frac{\pi_t(x)}{1 - \pi_t(\mathcal{X}^*)} \leq \frac{1}{2^{n-1} - 1}. \quad (3.51)$$

Let $\beta_t = 1/(2^{n-1} - 1)$, by Theorem 3.1, we have

$$\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] = \Omega(2^n). \quad (3.52)$$

□

3.3 Summary

In this chapter, we establish a bridge between two fundamental theoretical issues of EAs, that is, the expected running time and the convergence rate. With this bridge, we present the convergence-based analysis approach for analyzing the expected running time of EAs. As an illustrative example of applying convergence-based analysis, we prove the exponential lower bound of the expected running time for (1+1)-EA and RLS solving the constrained Trap problem.



Running Time Analysis: Switch Analysis

Previously introduced analysis approaches, i.e., fitness-level, drift analysis, and convergence-based analysis, provide paths, following which one can derive the expected running time of an EA solving an optimization problem from scratch. This chapter presents another approach, *switch analysis* [Yu et al., 2015], for analyzing the running time of EAs. Different from the previous approaches, switch analysis compares the expected running time of two EA processes. For analyzing a given target EA solving a given problem, switch analysis works by comparing its expected running time with that of a *reference* EA process, where the reference process can be specially designed to be somewhat similar to the given EA process but easier to be analyzed. We demonstrate the use of switch analysis by proving the expected running time lower bound of any mutation-based EA on the UBoolean function class defined in Definition 2.2, i.e., the pseudo-Boolean function class with a unique global optimal solution.

The rest of this chapter is organized as follows. Sections 4.1 and 4.2 present the switch analysis approach and its application illustration, respectively. Section 4.3 concludes this chapter.

4.1 Switch Analysis: Framework

Given two Markov chains $\xi \in \mathcal{X}$ and $\xi' \in \mathcal{Y}$, let τ and τ' denote their first hitting time, respectively. We present the switch analysis approach in Theorem 4.1 which works by comparing the DCFHT of the two chains, i.e., $\mathbb{E}[\tau | \xi_0 \sim \pi_0]$ and $\mathbb{E}[\tau' | \xi'_0 \sim \pi_0^\phi]$, where π_0 and π_0^ϕ are their initial state distributions, respectively. Considering that two chains may have different state spaces, we utilize *aligned mappings* in Definition 4.1.

Definition 4.1 (Aligned Mapping). *Given two spaces \mathcal{X} and \mathcal{Y} with target subspaces \mathcal{X}^* and \mathcal{Y}^* , respectively, a function $\phi : \mathcal{X} \rightarrow \mathcal{Y}$ is called*

(1) *a left-aligned mapping if $\forall x \in \mathcal{X}^* : \phi(x) \in \mathcal{Y}^*$;*

- (2) a right-aligned mapping if $\forall x \in \mathcal{X} \setminus \mathcal{X}^* : \phi(x) \notin \mathcal{Y}^*$;
(3) an optimal-aligned mapping if it is both left-aligned and right-aligned.

Note that the function $\phi : \mathcal{X} \rightarrow \mathcal{Y}$ implies that $\forall x \in \mathcal{X}$, there exists one and only one $y \in \mathcal{Y}$ such that $\phi(x) = y$, but it may not be an injective or surjective mapping. To simplify the notation, let the mapping $\phi^{-1}(y) = \{x \in \mathcal{X} \mid \phi(x) = y\}$ denote the *inverse solution set* of the function. Note that $\phi^{-1}(y)$ can be the empty set for some $y \in \mathcal{Y}$. The notation of ϕ is also extended to enable set input, i.e., $\forall X \subseteq \mathcal{X} : \phi(X) = \cup_{x \in X} \{\phi(x)\}$ and $\forall Y \subseteq \mathcal{Y} : \phi^{-1}(Y) = \cup_{y \in Y} \phi^{-1}(y)$. By the set extension, it holds that, if ϕ is a left-aligned mapping, $\mathcal{X}^* \subseteq \phi^{-1}(\mathcal{Y}^*)$; if ϕ is a right-aligned mapping, $\phi^{-1}(\mathcal{Y}^*) \subseteq \mathcal{X}^*$; if ϕ is an optimal-aligned mapping, $\mathcal{X}^* = \phi^{-1}(\mathcal{Y}^*)$.

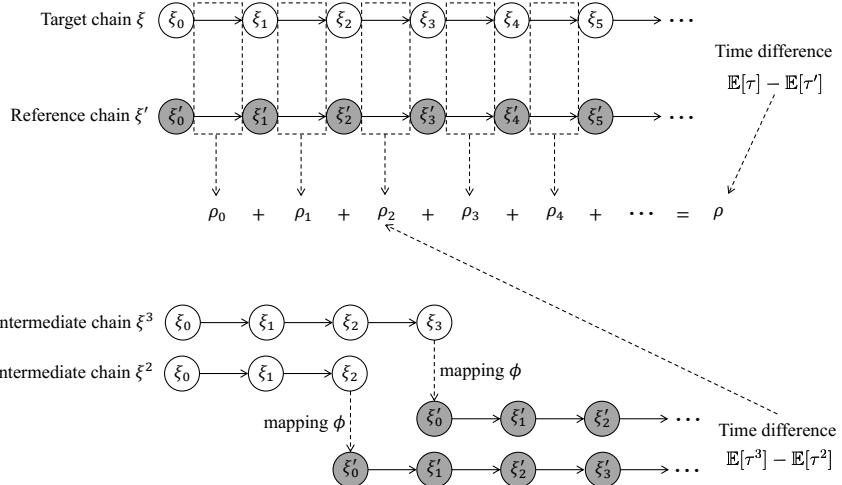


Figure 4.1. Intuition of the switch analysis approach. The time difference of two chains ξ and ξ' is obtained by summing up the time difference at each step of the two chains, calculated by the time difference of two intermediate chains ξ^i and ξ^{i+1} . Note that the two chains ξ^i and ξ^{i+1} are different only at time i .

The switch analysis approach presented in Theorem 4.1, is illustrated in Figure 4.1. The intuition is that, if we can bound the difference of the two chains on the one-step change of the DCFHT, i.e., the time difference of the two intermediate chains in Figure 4.1, we can obtain the difference of their DCFHT by summing up all one-step differences. We find that the calculation of the one-step difference can be drastically simplified by considering the one-step transitions of the two chains under the same distribution of one chain, i.e., π_t in Eq. (4.1), and on the same ground of CFHT of the other chain, i.e., $\mathbb{E}[\tau']$ in Eq. (4.1). The one-step differences, ρ_t , are then summed up to bound the difference of their DCFHT. The right (or left)-aligned mapping is used to allow the two chains to have different state spaces.

Theorem 4.1 (Switch Analysis). Given two absorbing Markov chains $\xi \in \mathcal{X}$ and $\xi' \in \mathcal{Y}$, let τ and τ' denote their first hitting time, respectively, and let π_t denote the distribution of ξ_t . Given a series of values $\{\rho_t \in \mathbb{R}\}_{t=0}^{+\infty}$ with $\rho = \sum_{t=0}^{+\infty} \rho_t$ and a right (or left)-aligned mapping $\phi : \mathcal{X} \rightarrow \mathcal{Y}$, if $\mathbb{E}[\tau | \xi_0 \sim \pi_0]$ is finite and

$$\begin{aligned} \forall t : & \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_t(x) P(\xi_{t+1} \in \phi^{-1}(y) | \xi_t = x) \mathbb{E}[\tau' | \xi'_0 = y] \\ & \leq (\text{or } \geq) \sum_{u, y \in \mathcal{Y}} \pi_t^\phi(u) P(\xi'_1 = y | \xi'_0 = u) \mathbb{E}[\tau' | \xi'_1 = y] + \rho_t, \end{aligned} \quad (4.1)$$

where $\pi_t^\phi(y) = \pi_t(\phi^{-1}(y)) = \sum_{x \in \phi^{-1}(y)} \pi_t(x)$, we have

$$\mathbb{E}[\tau | \xi_0 \sim \pi_0] \leq (\text{or } \geq) \mathbb{E}[\tau' | \xi'_0 \sim \pi_0^\phi] + \rho. \quad (4.2)$$

To prove the theorem, we define the intermediate Markov chain ξ^k for $k \in \{0, 1, \dots\}$. Denote the one-step transition of ξ as tr , and the one-step transition of ξ' as tr' . ξ^k is a Markov chain which

1. is initially in the state space \mathcal{X} and has the same initial state distribution as ξ , i.e., $\pi_0^k = \pi_0$;
2. uses the one-step transition tr at time $\{0, 1, \dots, k-1\}$ if $k > 0$, i.e., it is identical to the chain ξ at the first k steps;
3. switches to the state space \mathcal{Y} at time k , by mapping the distribution π_k of states over \mathcal{X} to the distribution π_k^ϕ of states over \mathcal{Y} via ϕ ;
4. uses the one-step transition tr' from time k , i.e., it then acts like the chain ξ' from time 0.

For the first hitting time τ^k of the intermediate Markov chain ξ^k , its first hitting event is counted as $\xi_t^k \in \mathcal{X}^*$ for $t = 0, 1, \dots, k-1$ and as $\xi_t^k \in \mathcal{Y}^*$ for $t \geq k$. Therefore, the first hitting events of ξ^0 and ξ^∞ are the same as that of ξ' and ξ , respectively. The expectation of τ^k is shown in Lemma 4.1, the proof of which is provided in Appendix A.1 of the book.

Lemma 4.1. Given two absorbing Markov chains $\xi \in \mathcal{X}$ and $\xi' \in \mathcal{Y}$, and a right-aligned mapping $\phi : \mathcal{X} \rightarrow \mathcal{Y}$, let τ and τ' denote the first hitting time of ξ and ξ' , respectively, and let π_t denote the distribution of ξ_t . For the first hitting time τ^k of the intermediate chain ξ^k with any $k \in \{0, 1, \dots\}$, we have

$$\begin{aligned} \mathbb{E}[\tau^k | \xi_0^k \sim \pi_0] &= k - \sum_{t=0}^{k-1} \pi_t(\mathcal{X}^*) \\ &+ \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_{k-1}(x) P(\xi_k \in \phi^{-1}(y) | \xi_{k-1} = x) \mathbb{E}[\tau' | \xi'_0 = y] \\ &- \sum_{x \in \mathcal{X}^*, y \in \mathcal{Y}} \pi_{k-1}(x) P(\xi_k \in \phi^{-1}(y) | \xi_{k-1} = x) \mathbb{E}[\tau' | \xi'_0 = y]. \end{aligned} \quad (4.3)$$

Proof of Theorem 4.1 (“ \leq ” case).

Firstly we prove the “ \leq ” case which requires a right-aligned mapping.

$\forall t < k$, because the chain ξ^k and ξ are identical before time k , we have $\pi_t = \pi_t^k$, and thus

$$\forall t < k : \pi_t^k(\mathcal{X}^*) = \pi_t(\mathcal{X}^*) \geq \pi_t(\phi^{-1}(\mathcal{Y}^*)) = \pi_t^\phi(\mathcal{Y}^*). \quad (4.4)$$

Note that ϕ is right-aligned and thus $\phi^{-1}(\mathcal{Y}^*) \subseteq \mathcal{X}^*$.

We prove the theorem by induction on the k of the intermediate Markov chain ξ^k .

(a) **Initialization** is to prove the case $k = 0$; this is trivial because $\xi^0 = \xi'$ and thus $\mathbb{E}[\tau^0 \mid \xi_0^0 \sim \pi_0] = \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi]$.

(b) **Inductive Hypothesis** assumes that $\forall k \leq K - 1$ ($K \geq 1$),

$$\mathbb{E}[\tau^k \mid \xi_0^k \sim \pi_0] \leq \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] + \sum_{t=0}^{k-1} \rho_t; \quad (4.5)$$

we are going to prove

$$\mathbb{E}[\tau^K \mid \xi_0^K \sim \pi_0] \leq \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] + \sum_{t=0}^{K-1} \rho_t. \quad (4.6)$$

Let $\Delta(K) = \sum_{x \in \mathcal{X}^*, y \in \mathcal{Y}} \pi_{K-1}(x) P(\xi_K \in \phi^{-1}(y) \mid \xi_{K-1} = x) \mathbb{E}[\tau' \mid \xi'_0 = y]$. Applying Lemma 4.1, we have

$$\begin{aligned} \mathbb{E}[\tau^K \mid \xi_0^K \sim \pi_0] &= K - \sum_{t=0}^{K-1} \pi_t(\mathcal{X}^*) \\ &\quad + \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_{K-1}(x) P(\xi_K \in \phi^{-1}(y) \mid \xi_{K-1} = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ &\quad - \sum_{x \in \mathcal{X}^*, y \in \mathcal{Y}} \pi_{K-1}(x) P(\xi_K \in \phi^{-1}(y) \mid \xi_{K-1} = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ &\leq K - \sum_{t=0}^{K-1} \pi_t(\mathcal{X}^*) + \rho_{K-1} - \Delta(K) \\ &\quad + \sum_{u, y \in \mathcal{Y}} \pi_{K-1}^\phi(u) P(\xi'_1 = y \mid \xi'_0 = u) \mathbb{E}[\tau' \mid \xi'_1 = y] \end{aligned} \quad (4.7)$$

$$\begin{aligned} &\leq K - \sum_{t=0}^{K-2} \pi_t(\mathcal{X}^*) - \pi_{K-1}^\phi(\mathcal{Y}^*) + \rho_{K-1} - \Delta(K) \\ &\quad + \sum_{u, y \in \mathcal{Y}} \pi_{K-1}^\phi(u) P(\xi'_1 = y \mid \xi'_0 = u) \mathbb{E}[\tau' \mid \xi'_1 = y], \end{aligned} \quad (4.8)$$

where Eq. (4.7) holds by Eq. (4.1) and the definition of $\Delta(K)$, and Eq. (4.8) holds by Eq. (4.4). Meanwhile, by Lemma 4.1, we have

$$\begin{aligned}
& \mathbb{E}[\tau^{K-1} \mid \xi_0^{K-1} \sim \pi_0] \\
&= (K-1) - \sum_{t=0}^{K-2} \pi_t(\mathcal{X}^*) - \Delta(K-1) \\
&\quad + \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_{K-2}(x) P(\xi_{K-1} \in \phi^{-1}(y) \mid \xi_{K-2} = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\
&= (K-1) - \sum_{t=0}^{K-2} \pi_t(\mathcal{X}^*) - \Delta(K-1) + \sum_{y \in \mathcal{Y}} \pi_{K-1}^\phi(y) \mathbb{E}[\tau' \mid \xi'_0 = y] \quad (4.9)
\end{aligned}$$

$$\begin{aligned}
&= K - \sum_{t=0}^{K-2} \pi_t(\mathcal{X}^*) - \pi_{K-1}^\phi(\mathcal{Y}^*) - \Delta(K-1) \\
&\quad + \sum_{u, y \in \mathcal{Y}} \pi_{K-1}^\phi(u) P(\xi'_1 = y \mid \xi'_0 = u) \mathbb{E}[\tau' \mid \xi'_1 = y], \quad (4.10)
\end{aligned}$$

where Eqs. (4.9) and (4.10) hold by Lemma 2.2. Substituting Eq. (4.10) into Eq. (4.8), we have

$$\begin{aligned}
\mathbb{E}[\tau^K \mid \xi_0^K \sim \pi_0] &\leq \mathbb{E}[\tau^{K-1} \mid \xi_0^{K-1} \sim \pi_0] + \rho_{K-1} + \Delta(K-1) - \Delta(K) \\
&\leq \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] + \sum_{t=0}^{K-1} \rho_t + \Delta(K-1) - \Delta(K), \quad (4.11)
\end{aligned}$$

where Eq. (4.11) holds by inductive hypothesis. Considering our definition of absorption that $\forall x \in \mathcal{X}^* : P(\xi_{t+1} \neq x \mid \xi_t = x) = 0$, we have

$$\forall x \in \mathcal{X}^* : \pi_{K-1}(x) \geq \pi_{K-2}(x), \quad (4.12)$$

$$\Delta(K) = \sum_{x \in \mathcal{X}^*} \pi_{K-1}(x) \mathbb{E}[\tau' \mid \xi'_0 = \phi(x)], \quad (4.13)$$

leading to

$$\Delta(K-1) - \Delta(K) = \sum_{x \in \mathcal{X}^*} (\pi_{K-2}(x) - \pi_{K-1}(x)) \mathbb{E}[\tau' \mid \xi'_0 = \phi(x)] \leq 0. \quad (4.14)$$

Thus, Eq. (4.11) results in Eq. (4.6).

(c) Concluding from (a) and (b), it holds that

$$\mathbb{E}[\tau^\infty \mid \xi_0^\infty \sim \pi_0] \leq \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] + \sum_{t=0}^{+\infty} \rho_t. \quad (4.15)$$

Because $\mathbb{E}[\tau \mid \xi_0 \sim \pi_0]$ is finite, $\mathbb{E}[\tau^\infty \mid \xi_0^\infty \sim \pi_0] = \mathbb{E}[\tau \mid \xi_0 \sim \pi_0]$. Finally, by $\rho = \sum_{t=0}^{+\infty} \rho_t$, we have

$$\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \leq \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] + \rho. \quad (4.16)$$

□

The “ \geq ” case requires a left-aligned mapping. Its proof is similar to that of the “ \leq ” case and is provided in Appendix A.1. Though the theorem is proved by treating the state spaces \mathcal{X} and \mathcal{Y} as discrete spaces, we can show that the theorem still holds if \mathcal{X} and/or \mathcal{Y} are/is continuous, by replacing the sum over the state space with the integral, which does not affect the inductive proof. We present only the discrete space version as only discrete optimization is studied.

Using the theorem to compare two chains, we can waive the long-term behavior of one chain, as Eq. (4.1) does not involve the term $\mathbb{E}[\tau \mid \xi_0 = x]$. Therefore, the theorem can simplify the analysis of an EA by comparing it with an easy-to-analyze one.

When the Markov chain $\xi' \in \mathcal{Y}$ is homogeneous, i.e., the transition is static regardless of time, we can have a compact expression of the switch analysis approach, by rewriting Eq. (4.1) as

$$\begin{aligned} \forall t : \sum_{y \in \mathcal{Y}} \mathbb{E}[\tau' \mid \xi'_0 = y] \cdot & (P(\phi(\xi_{t+1}) = y \mid \xi_t \sim \pi_t) - P(\xi'_{t+1} = y \mid \xi'_t \sim \pi_t^\phi)) \\ \leq (\text{or } \geq) \rho_t. \end{aligned} \quad (4.17)$$

By this expression, we can interpret that ρ_t bounds the sum of the weighted distribution difference of the two intermediate chains ξ^{t+1} and ξ^t at time $t + 1$, where the weight is given by the CFHT of the chain ξ' . Because ξ^{t+1} and ξ^t are different only at time t , ρ_t actually bounds the difference of using transition tr and tr' at time t . Thus, ρ , i.e., the difference of the DCFHT of the original two chains ξ and ξ' , is $\sum_{t=0}^{+\infty} \rho_t$, i.e., the sum of the difference of using tr and tr' at each time.

4.2 Switch Analysis: Application Illustration

In this section, we give an example of applying the switch analysis approach. Algorithm 4.1 presents a general scheme of mutation-based EAs. It abstracts general population-based EAs employing only mutation operators, including many variants as introduced in [Sudholt, 2013, Witt, 2013].

In the following, we use switch analysis to prove that the expected running time of any mutation-based EA on the UBoolean function class in Definition 2.2 is no less than that of $(1+1)\text{-EA}_\mu$, i.e., Algorithm 4.2, on the One-Max problem in Definition 2.3. Note that $(1+1)\text{-EA}_\mu$ is the same as $(1+1)\text{-EA}$ except the initialization strategy: $(1+1)\text{-EA}_\mu$ selects the best from μ random solutions, whereas $(1+1)\text{-EA}$ directly uses one random solution. By using the fitness level method, Sudholt [2013] proved that the expected running time of arbitrary mutation-based EA with the mutation probability $1/n$ on UBoolean is no less than that of $(1+1)\text{-EA}_\mu$ on OneMax. Our result here generalizes to any mutation probability $p \in (0, 0.5)$. Witt [2013] proved the same result with stochastic dominance using drift analysis.

Algorithm 4.1 Scheme of A Mutation-based EA

Input: pseudo-Boolean function $f : \{0, 1\}^n \rightarrow \mathbb{R}$; positive integer $\mu \in \mathbb{N}^+$ **Output:** set of solutions from $\{0, 1\}^n$ **Process:**

- 1: select μ solutions $s_1, \dots, s_\mu \in \{0, 1\}^n$ uniformly at random;
 - 2: let $t = \mu$, and select a parent solution s from $\{s_1, \dots, s_t\}$ according to t and $f(s_1), \dots, f(s_t)$;
 - 3: **while** criterion is not met **do**
 - 4: apply bit-wise mutation on s to generate s_{t+1} ;
 - 5: select a parent solution s from $\{s_1, \dots, s_{t+1}\}$ according to $(t + 1)$ and $f(s_1), \dots, f(s_{t+1})$;
 - 6: $t = t + 1$
 - 7: **end while**
 - 8: **return** solutions from $\{s_1, \dots, s_t\}$ according to some strategy
-

Algorithm 4.2 (1+1)-EA $_\mu$ Algorithm

Input: pseudo-Boolean function $f : \{0, 1\}^n \rightarrow \mathbb{R}$; positive integer $\mu \in \mathbb{N}^+$ **Output:** solution from $\{0, 1\}^n$ **Process:**

- 1: select μ solutions $s_1, \dots, s_\mu \in \{0, 1\}^n$ uniformly at random;
 - 2: let s = the best one among s_1, \dots, s_μ ;
 - 3: **while** criterion is not met **do**
 - 4: apply bit-wise mutation on s to generate s' ;
 - 5: **if** $f(s') \geq f(s)$ **then**
 - 6: $s = s'$
 - 7: **end if**
 - 8: **end while**
 - 9: **return** s
-

We give some lemmas which will be used in the following analysis. Their proofs are provided in Appendix A.1. Because the bits of OneMax are independent and their weights are the same, the CFHT $\mathbb{E}[\tau' | \xi'_t = x]$ of the Markov chain modeling (1+1)-EA solving OneMax depends only on $|x|_0$. Note that a state x is just a solution here. Let $\mathbb{E}(j)$ denote the CFHT $\mathbb{E}[\tau' | \xi'_t = x]$ with $|x|_0 = j$. $\mathbb{E}(0) = 0$ implies the optimal solution. Lemma 4.2 gives the order on $\mathbb{E}(j)$, disclosing that $\mathbb{E}(j)$ increases with j .

Lemma 4.2. *For any mutation probability $p \in (0, 0.5)$, it holds that*

$$\mathbb{E}(0) < \mathbb{E}(1) < \mathbb{E}(2) < \dots < \mathbb{E}(n). \quad (4.18)$$

Lemma 4.3 says that mutating a parent solution with a small number of 0-bits is more likely to generate an offspring solution with a small number of 0-bits. Note that we consider $|\cdot|_0$ instead of $|\cdot|_1$ in [Witt, 2013]. It still holds due to symmetry.

Lemma 4.3. [Witt, 2013] Let $s, s' \in \{0, 1\}^n$ be two search points satisfying $|s|_0 < |s'|_0$. Denote by $\text{mut}(s)$ the random string obtained by mutating each bit of s independently with probability p . Let j be an arbitrary integer in $[n]$. If $p \leq 0.5$, it holds that

$$\mathbb{P}(|\text{mut}(s)|_0 \leq j) \geq \mathbb{P}(|\text{mut}(s')|_0 \leq j). \quad (4.19)$$

Lemma 4.4 is a generalization of Lemma 3.2.

Lemma 4.4. Let m ($m \geq 1$) be an integer. If it satisfies that

$$(1) \quad 0 \leq E_0 < E_1 < \cdots < E_m, \quad (4.20)$$

$$(2) \quad \forall 0 \leq i \leq m : P_i, Q_i \geq 0 \wedge \sum_{i=0}^m P_i = \sum_{i=0}^m Q_i = 1, \quad (4.21)$$

$$(3) \quad \forall 0 \leq k \leq m-1 : \sum_{i=0}^k P_i \leq \sum_{i=0}^k Q_i, \quad (4.22)$$

then it holds that

$$\sum_{i=0}^m P_i \cdot E_i \geq \sum_{i=0}^m Q_i \cdot E_i. \quad (4.23)$$

Theorem 4.2. The expected running time of any mutation-based EA with μ initial solutions and any mutation probability $p \in (0, 0.5)$ on UBoolean is no less than that of (1+1)-EA $_\mu$ with the same p on the OneMax problem.

Proof. We use a history-encoded Markov chain to model a mutation-based EA in Algorithm 4.1. Let $\mathcal{X} = \{(s_1, \dots, s_t) \mid s_j \in \{0, 1\}^n, t \geq \mu\}$, where (s_1, \dots, s_t) is a sequence of solutions which is the search history of the EA until time t and μ is the number of initial solutions; $\mathcal{X}^* = \{x \in \mathcal{X} \mid 1^n \in x\}$, where $s \in x$ means that s appears in the sequence. The chain $\xi \in \mathcal{X}$ models an arbitrary mutation-based EA on any function in UBoolean. It holds that $\forall i \geq 0 : \xi_i \in \{(s_1, \dots, s_t) \mid s_j \in \{0, 1\}^n, t = \mu + i\}$.

Let $\xi' \in \mathcal{Y}$ model a reference process of (1+1)-EA running on the OneMax problem. $\mathcal{Y} = \{0, 1\}^n$ and $\mathcal{Y}^* = \{1^n\}$. We construct the function $\phi : \mathcal{X} \rightarrow \mathcal{Y}$ as $\phi(x) = 1^{n-i}0^i$ with $i = \min\{|s|_0 \mid s \in x\}$. Such a ϕ is an optimal-aligned mapping because $\phi(x) = 1^n$ iff $1^n \in x$, i.e., $x \in \mathcal{X}^*$.

First, we examine the condition, i.e., Eq. (4.1), of switch analysis. $\forall x \notin \mathcal{X}^*$, assume $|\phi(x)|_0 = i > 0$. Let P_j be the probability for the offspring solution generated on $\phi(x)$ by bit-wise mutation to have j number of 0-bits. For ξ' , it accepts only the offspring solution which does not have more 0-bits than the parent solution, and thus, we have

$$\sum_{y \in \mathcal{Y}} \mathbb{P}(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] = \sum_{j=0}^{i-1} P_j \mathbb{E}(j) + \left(1 - \sum_{j=0}^{i-1} P_j\right) \mathbb{E}(i). \quad (4.24)$$

For ξ , it selects a solution s from x for reproduction. Let P'_j be the probability for the offspring solution s' generated on s by bit-wise mutation to have j number of 0-bits. If $|s'|_0 < i$, $|\phi((x, s'))|_0 = |s'|_0$; otherwise, $|\phi((x, s'))|_0 = i$, where (x, s') is the solution sequence until time $t + 1$. Thus,

$$\sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] = \sum_{j=0}^{i-1} P'_j \mathbb{E}(j) + \left(1 - \sum_{j=0}^{i-1} P'_j\right) \mathbb{E}(i). \quad (4.25)$$

By the definition of ϕ , we have $|s|_0 \geq |\phi(x)|_0 = i$. Then, by Lemma 4.3, we have $\forall k \in [n] : \sum_{j=0}^k P_j \geq \sum_{j=0}^k P'_j$. Meanwhile, $\mathbb{E}(i)$ increases with i as shown in Lemma 4.2. Thus, by Lemma 4.4, we have

$$\sum_{j=0}^{i-1} P'_j \mathbb{E}(j) + \left(1 - \sum_{j=0}^{i-1} P'_j\right) \mathbb{E}(i) \geq \sum_{j=0}^{i-1} P_j \mathbb{E}(j) + \left(1 - \sum_{j=0}^{i-1} P_j\right) \mathbb{E}(i), \quad (4.26)$$

which is equivalent to

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ & \geq \sum_{y \in \mathcal{Y}} P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y]. \end{aligned} \quad (4.27)$$

In other words, the condition, i.e., Eq. (4.1), of switch analysis holds with $\rho_t = 0$, implying $\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \geq \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi]$.

Next, we examine $\mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi]$. For mutation-based EAs shown in Algorithm 4.1, the initial population consists of μ solutions s_1, \dots, s_μ randomly selected from $\{0, 1\}^n$. By the definition of ϕ , $\forall 0 \leq j \leq n : \pi_0^\phi(\{y \in \mathcal{Y} \mid |y|_0 = j\})$ is the probability of $\min\{|s_1|_0, \dots, |s_\mu|_0\} = j$. Thus,

$$\begin{aligned} \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] &= \sum_{j=0}^n \pi_0^\phi(\{y \in \mathcal{Y} \mid |y|_0 = j\}) \mathbb{E}(j) \\ &= \sum_{j=0}^n P(\min\{|s_1|_0, \dots, |s_\mu|_0\} = j) \mathbb{E}(j), \end{aligned} \quad (4.28)$$

which is actually the DCFHT of the Markov chain modeling (1+1)-EA $_\mu$ solving the OneMax problem.

Because both mutation-based EAs and (1+1)-EA $_\mu$ evaluate μ solutions in initialization and evaluate one solution in each iteration, $\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \geq \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi]$ implies that the expected running time of any mutation-based EA on UBoolean is no less than that of (1+1)-EA $_\mu$ on OneMax. \square

4.3 Summary

In this chapter, we present the switch analysis approach for analyzing the expected running time of EAs. Switch analysis works by comparing two EA processes. Through the comparison, we can avoid the obstacle of analyzing the long-term behavior of a complicated process, and only need to compare one-step transition probabilities of the two processes. This enables to study the expected running time of a complicated target process by considering a relatively simpler reference process. For an example of applying switch analysis, we prove the expected running time lower bound of mutation-based EAs on the UBoolean function class.



5

Running Time Analysis: Comparison and Unification

As we have introduced several approaches, i.e., fitness level, drift analysis, convergence-based analysis and switch analysis, for running time analysis of EAs, a natural question is how different *analysis approaches* relate to each other. To address this question, this chapter presents formal characterization of these analysis approaches, and their *reducibility* analyses [Yu et al., 2015, Yu and Qian, 2015]. Roughly speaking, an approach \mathfrak{A}_1 is *reducible* to \mathfrak{A}_2 if \mathfrak{A}_2 can derive at least the same tight bound as \mathfrak{A}_1 while requiring no more information, implying that \mathfrak{A}_2 is at least as powerful as \mathfrak{A}_1 . We will show that fitness level, drift analysis, and convergence-based analysis are all reducible to switch analysis. Meanwhile, we find that switch analysis is not reducible to fitness level. We compare switch analysis with drift analysis on analyzing (1+1)-EA solving the discrete linear problem, and derive a new upper bound of its running time by switch analysis. We also compare switch analysis with convergence-based analysis on analyzing (1+1)-EA and RLS solving the constrained Trap problem, and find that switch analysis can derive lower bounds of the running time tighter than the known ones obtained by convergence-based analysis. These results not only disclose the power of switch analysis, but also provide a unified view of different running time analysis approaches for EAs.

The rest of this chapter is organized as follows. Section 5.1 introduces formalization and reducibility of analysis approaches. Sections 5.2 to 5.4 present the reducibility analyses between switch analysis and the other three approaches, i.e., fitness level, drift analysis, and convergence-based analysis, respectively. Section 5.5 discusses unification of analysis approaches. Section 5.6 concludes this chapter.

5.1 Analysis Approaches: Formalization

Analysis approaches, in general, were usually conceptually described and applied to problems case by case. A general analysis approach for EA pro-

cesses commonly specifies a set of variables to examine and a procedure to follow. Therefore, in this context, we propose to treat an analysis approach like an algorithm with input, parameters, and output. The input is some structure information about the concerned EA process such as the state transition probabilities; the parameters are ad hoc information used by this analysis approach, such as information about the reference process of switch analysis; the output is a lower and/or upper bound of the running time, as described in Definition 5.1. The state space \mathcal{X} itself of the EA process is not regarded as a part of the input, because it can be known ahead by specifying the optimization problem. For an analysis approach \mathfrak{A} , let $\mathfrak{A}^l(\mathcal{I}; \mathcal{P})$ and $\mathfrak{A}^u(\mathcal{I}; \mathcal{P})$ denote the lower and upper bounds, respectively, given the input \mathcal{I} and parameters \mathcal{P} . When the context is clear, \mathcal{I} and \mathcal{P} will be omitted.

Definition 5.1 (EA Analysis Approach). *A procedure \mathfrak{A} is called an EA analysis approach if for any EA process $\xi \in \mathcal{X}$ with initial state ξ_0 and transition probability matrix \mathbf{P} , given $\mathcal{I} = g(\xi_0, \mathbf{P})$ for some function g and a set of parameters $\mathcal{P}(\mathcal{I})$, \mathfrak{A} outputs a lower running time bound of ξ notated as $\mathfrak{A}^l(\mathcal{I}; \mathcal{P})$ and/or an upper bound $\mathfrak{A}^u(\mathcal{I}; \mathcal{P})$.*

As for the formal characterization of switch analysis in Characterization 5.1, all variables derived from the reference process are regarded as parameters, including bounds of the one-step transition probabilities and the CFHT of the reference process. The input of switch analysis includes bounds of the one-step transition probabilities of the concerned EA process. It should be noted that the tightness of the input bounds determines the tightness of the output bounds one can have, and the goodness of the selected parameter values determines how close the actually derived bounds are to the optimal bounds.

Characterization 5.1 (Switch Analysis). *For an EA process $\xi \in \mathcal{X}$, the switch analysis approach \mathfrak{A}_{SA} is defined by:*

Parameters: *a reference process $\xi' \in \mathcal{Y}$ with bounds of its transition probabilities $P(\xi'_1 | \xi'_0)$ and CFHT $\mathbb{E}[\tau' | \xi'_t = y] \forall y \in \mathcal{Y}, t \in \{0, 1\}$, and a right-aligned mapping $\phi^u : \mathcal{X} \rightarrow \mathcal{Y}$ or a left-aligned mapping $\phi^l : \mathcal{X} \rightarrow \mathcal{Y}$.*

Input: *bounds of one-step transition probabilities $P(\xi_{t+1} | \xi_t)$.*

Output: *denoting $\pi_t^\phi(y) = \pi_t(\phi^{-1}(y)) \forall y \in \mathcal{Y}$,*

$$\mathfrak{A}_{SA}^u = \mathbb{E}[\tau' | \xi'_0 \sim \pi_0^\phi] + \rho^u \text{ where } \rho^u = \sum_{t=0}^{+\infty} \rho_t^u \text{ and}$$

$$\forall t : \rho_t^u \geq \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_t(x) P(\xi_{t+1} \in \phi^{-1}(y) | \xi_t = x) \mathbb{E}[\tau' | \xi'_0 = y] -$$

$$\sum_{u, y \in \mathcal{Y}} \pi_t^\phi(u) P(\xi'_1 = y | \xi'_0 = u) \mathbb{E}[\tau' | \xi'_1 = y],$$

$$\mathfrak{A}_{SA}^l = \mathbb{E}[\tau' | \xi'_0 \sim \pi_0^\phi] + \rho^l \text{ where } \rho^l = \sum_{t=0}^{+\infty} \rho_t^l \text{ and}$$

$$\forall t : \rho_t^l \leq \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_t(x) P(\xi_{t+1} \in \phi^{-1}(y) | \xi_t = x) \mathbb{E}[\tau' | \xi'_0 = y] -$$

$$\sum_{u, y \in \mathcal{Y}} \pi_t^\phi(u) P(\xi'_1 = y | \xi'_0 = u) \mathbb{E}[\tau' | \xi'_1 = y].$$

As analysis approaches are characterized by their input, parameters and output, we can then study their relative strength. In the first thought, if one

approach always derives tighter running time bounds than another, the former is more powerful. The tightness, however, is affected by many aspects. Different usages of an approach can result in different bounds. We shall not compare the results of particular uses of two approaches. Therefore, we define the reducibility between analysis approaches in Definition 5.2.

Definition 5.2 (Reducible). *For two EA analysis approaches \mathfrak{A}_1 and \mathfrak{A}_2 , if for any input \mathcal{I} and parameters \mathcal{P}_1 , there exists a transformation T and parameters \mathcal{P}_2 , possibly depending on \mathcal{P}_1 , such that*

- (1) $\mathfrak{A}_1^u(\mathcal{I}; \mathcal{P}_1) \geq \mathfrak{A}_2^u(T(\mathcal{I}); \mathcal{P}_2)$, then \mathfrak{A}_1 is upper-bound reducible to \mathfrak{A}_2 ;
- (2) $\mathfrak{A}_1^l(\mathcal{I}; \mathcal{P}_1) \leq \mathfrak{A}_2^l(T(\mathcal{I}); \mathcal{P}_2)$, then \mathfrak{A}_1 is lower-bound reducible to \mathfrak{A}_2 .

Moreover, \mathfrak{A}_1 is reducible to \mathfrak{A}_2 if it is both upper-bound reducible and lower-bound reducible to \mathfrak{A}_2 .

By the definition, for two analysis approaches \mathfrak{A}_1 and \mathfrak{A}_2 , \mathfrak{A}_1 is said to be reducible to \mathfrak{A}_2 if it is possible to construct an input of \mathfrak{A}_2 by the transformation T solely from the input of \mathfrak{A}_1 , while \mathfrak{A}_2 with some parameters can output a bound which is at least as good as that of \mathfrak{A}_1 . If no such transformation or parameter exists, \mathfrak{A}_1 is said to be not reducible to \mathfrak{A}_2 . Intuitively, there are two possible reasons that one approach is not reducible to the other: (1) the latter cannot take all the input of the former, i.e., T has to suffer from the missing of important information in the input; (2) although T does not miss information, the latter cannot make full use of it. When \mathfrak{A}_1 is proved to be reducible to \mathfrak{A}_2 , one can say that \mathfrak{A}_2 is at least as powerful as \mathfrak{A}_1 . Note that this does not imply that \mathfrak{A}_2 is easier than \mathfrak{A}_1 to use. The usability of an analysis approach also depends on its intuitiveness and the background of an analyst.

In the following, we will use reducibility to compare switch analysis with fitness level, drift analysis and convergence-based analysis, respectively.

5.2 Switch Analysis vs. Fitness Level

We first present the characterization of the fitness level method, then prove that the fitness level method is reducible to switch analysis, and finally show that switch analysis is not reducible to the fitness level method at least on the Peak problem.

Fitness level methods as presented in Theorems 2.1 and 2.2 are used for analyzing the running time of elitist EAs. The main idea is to divide the solution space into levels according to their fitness values and then estimate the transition probabilities of jumping to higher levels. The estimation of the probabilities leads to the estimation of the expected running time. As the original fitness level method in Theorem 2.1 is a special case of the refined one in Theorem 2.2 in respect of upper and lower bounds, we characterize the fitness level method using the latter theorem in Characterization 5.2.

Characterization 5.2 (Fitness Level). For an EA process $\xi \in \mathcal{X}$, the fitness level method \mathfrak{A}_{FL} is defined by:

Parameters: a $<_f$ -partition $\{\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_m\}$, where $\mathcal{S}_1 <_f \mathcal{S}_2 <_f \dots <_f \mathcal{S}_m = \mathcal{S}^*$.

Input: for some non-negative variables $\sum_{j=i+1}^m \gamma_{i,j} = 1$, transition probability bounds

$$v_i \leq \min_{x \in \mathcal{S}_i} \min_j \frac{1}{\gamma_{i,j}} P(\xi_{t+1} \in \mathcal{S}_j | \xi_t = x),$$

$$u_i \geq \max_{x \in \mathcal{S}_i} \max_j \frac{1}{\gamma_{i,j}} P(\xi_{t+1} \in \mathcal{S}_j | \xi_t = x),$$

$$\chi_u \geq \gamma_{i,j} / \sum_{k=j}^m \gamma_{i,k} \geq \chi_l \quad \forall i < j < m,$$

$$\chi_u \geq 1 - v_{j+1}/v_j \text{ and } \chi_l \geq 1 - u_{j+1}/u_j \quad \forall 1 \leq j \leq m-2.$$

Output:

$$\mathfrak{A}_{FL}^u = \sum_{i=1}^{m-1} \pi_0(\mathcal{S}^i) \cdot \left(\frac{1}{v_i} + \chi_u \sum_{j=i+1}^{m-1} \frac{1}{v_j} \right),$$

$$\mathfrak{A}_{FL}^l = \sum_{i=1}^{m-1} \pi_0(\mathcal{S}^i) \cdot \left(\frac{1}{u_i} + \chi_l \sum_{j=i+1}^{m-1} \frac{1}{u_j} \right).$$

Theorem 5.1. \mathfrak{A}_{FL} is reducible to \mathfrak{A}_{SA} .

Before proving the theorem, we introduce a simple Markov chain, i.e., OneJump, which will be used as the reference chain for switch analysis.

Definition 5.3 (OneJump). The OneJump chain with state space dimension n is a homogeneous Markov chain $\xi \in \{0, 1\}^n$ with $(n + 1)$ parameters $\{p_0, \dots, p_n\}$, $p_i \in [0, 1]$, and target state 1^n . Its transition probability is defined as, $\forall t \geq 0, x \in \{0, 1\}^n$:

$$P(\xi_{t+1} = y | \xi_t = x) = \begin{cases} p_{|x|_1}, & y = 1^n; \\ 1 - p_{|x|_1}, & y = x; \\ 0, & \text{otherwise.} \end{cases} \quad (5.1)$$

It is straightforward to calculate the CFHT of OneJump: $1/p_{n-j}$ when starting from a state x with $|x|_1 = n - j$. As the CFHT depends only on the number of 0-bits, let $\mathbb{E}_{oj}(j) = 1/p_{n-j}$ denote the CFHT of OneJump starting from a state with j 0-bits, for simplicity.

Theorem 5.1 is proved by combining Lemmas 5.1 and 5.2, which respectively prove the upper-bound and lower-bound reducibility.

Lemma 5.1. \mathfrak{A}_{FL} is upper-bound reducible to \mathfrak{A}_{SA} .

Proof. The proof is by finding the parameters and input of \mathfrak{A}_{SA} from that of \mathfrak{A}_{FL} , and showing that $\mathfrak{A}_{SA}^u \leq \mathfrak{A}_{FL}^u$. Denote $\xi \in \mathcal{X}$ as the EA process to be analyzed. By the parameter of \mathfrak{A}_{FL} , we have a $<_f$ -partition $\{\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_m\}$, which divides the solution space into m subspaces. We know some variables v_i , $\gamma_{i,j}$ and χ_u in Characterization 5.2 as the input of \mathfrak{A}_{FL} .

Let the reference process $\xi' \in \mathcal{Y} = \{0, 1\}^{m-1}$ be the OneJump chain with dimension $(m - 1)$ and parameters $p_i = 1/(\frac{1}{v_{i+1}} + \chi_u \sum_{j=i+2}^{m-1} \frac{1}{v_j}) \quad \forall 0 \leq i < m - 1$. We construct the mapping function $\phi : \mathcal{X} \rightarrow \mathcal{Y}$ as $\forall x \in \mathcal{S}_i : \phi(x) = 1^{i-1}0^{m-i}$, which is optimal-aligned because $\phi(x) \in \mathcal{Y}^* = \{1^{m-1}\}$ iff $x \in \mathcal{S}_m$.

We then calculate the upper bound output of \mathfrak{A}_{SA} using the input of \mathfrak{A}_{FL} and the reference process. By the function ϕ , any population in the partition S_i is mapped to the state $1^{i-1}0^{m-i}$ for the reference process, starting from which OneJump takes $\mathbb{E}_{oj}(m-i)$ steps to reach its target state. We then consider one-step transition of ξ . By the input of \mathfrak{A}_{FL} , we know that variables v_i , together with $\gamma_{i,j}$ and χ_u , bound from below the probability of generating better solutions. Thus, we can have an upper bound of the left part of Eq. (4.1): for any non-optimal state $x \in S_i$ with $i < m$,

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ & \leq v_i \sum_{j=i+1}^m \gamma_{i,j} \mathbb{E}_{oj}(m-j) + (1 - v_i) \mathbb{E}_{oj}(m-i), \end{aligned} \quad (5.2)$$

where the first part is the sum of the events with better solutions found and the second part is the remaining event. Eq. (5.2) holds because $\forall j \in \{i+1, \dots, m\} : \mathbb{E}_{oj}(m-i) \geq \mathbb{E}_{oj}(m-j)$, led by $\chi_u \geq 1 - v_{j+1}/v_j$. Meanwhile, considering the reference process, $\forall x \in S_i : \phi(x) = 1^{i-1}0^{m-i}$, and thus,

$$\sum_{y \in \mathcal{Y}} P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] = \mathbb{E}_{oj}(m-i) - 1, \quad (5.3)$$

where the equality holds by Lemma 2.1. Through comparing Eqs. (5.2) and (5.3), we have, $\forall x \in S_i$ with $i < m$,

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ & - \sum_{y \in \mathcal{Y}} P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] \\ & \leq 1 + \sum_{j=i+1}^{m-1} v_i \gamma_{i,j} \mathbb{E}_{oj}(m-j) - v_i \mathbb{E}_{oj}(m-i) \\ & = 1 + \sum_{j=i+1}^{m-1} v_i \gamma_{i,j} \left(\frac{1}{v_j} + \chi_u \sum_{k=j+1}^{m-1} \frac{1}{v_k} \right) - v_i \left(\frac{1}{v_i} + \chi_u \sum_{j=i+1}^{m-1} \frac{1}{v_j} \right) \end{aligned} \quad (5.4)$$

$$\begin{aligned} & = 1 + v_i \sum_{j=i+1}^{m-1} \frac{1}{v_j} \left(\gamma_{i,j} + \chi_u \sum_{k=i+1}^{j-1} \gamma_{i,k} \right) - v_i \left(\frac{1}{v_i} + \chi_u \sum_{j=i+1}^{m-1} \frac{1}{v_j} \right) \\ & \leq 1 + v_i \sum_{j=i+1}^{m-1} \frac{1}{v_j} \left(\chi_u \sum_{k=j}^m \gamma_{i,k} + \chi_u \sum_{k=i+1}^{j-1} \gamma_{i,k} \right) - v_i \left(\frac{1}{v_i} + \chi_u \sum_{j=i+1}^{m-1} \frac{1}{v_j} \right) \end{aligned} \quad (5.5)$$

$$= 0, \quad (5.6)$$

where Eq. (5.4) holds by $\mathbb{E}_{oj}(m - j) = 1/p_{j-1}$ and $\mathbb{E}_{oj}(m - i) = 1/p_{i-1}$, Eq. (5.5) holds by $\chi_u \geq \gamma_{i,j} / \sum_{k=j}^m \gamma_{i,k}$, and Eq. (5.6) holds by $\sum_{k=i+1}^m \gamma_{i,k} = 1$. Thus, we have, $\forall t \geq 0$,

$$\begin{aligned} & \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_t(x) P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ & - \sum_{u, y \in \mathcal{Y}} \pi_t^\phi(u) P(\xi'_1 = y \mid \xi'_0 = u) \mathbb{E}[\tau' \mid \xi'_1 = y] \leq 0. \end{aligned} \quad (5.7)$$

Therefore, $\forall t : \rho_t^u = 0$ is a proper assignment of ρ_t^u in Characterization 5.1, and thus, $\rho^u = 0$. We can then calculate the upper bound output. Considering $\pi_0^\phi(y) = \pi_0(\phi^{-1}(y))$, we have

$$\begin{aligned} \mathfrak{A}_{SA}^u &= \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] + 0 = \sum_{i=1}^m \pi_0(\mathcal{S}_i) \mathbb{E}_{oj}(m - i) \\ &= \sum_{i=1}^{m-1} \pi_0(\mathcal{S}_i) \left(\frac{1}{v_i} + \chi_u \sum_{j=i+1}^{m-1} \frac{1}{v_j} \right) = \mathfrak{A}_{FL}^u, \end{aligned} \quad (5.8)$$

which proves the lemma. \square

Lemma 5.2. \mathfrak{A}_{FL} is lower-bound reducible to \mathfrak{A}_{SA} .

The proof of Lemma 5.2 is similar to that of Lemma 5.1 except the change of variables and the corresponding inequality directions.

The proof of the reducibility is constructive, providing a way for switch analysis to simulate the fitness level method. Through the way, any proofs accomplished using the fitness level method can also be accomplished using switch analysis.

As we have proved that \mathfrak{A}_{FL} is reducible to \mathfrak{A}_{SA} , a following natural question is whether the inverse holds, i.e., “is \mathfrak{A}_{SA} reducible to \mathfrak{A}_{FL} ?” Through the Peak problem, we show that the answer to the question is negative, implying that switch analysis is strictly more powerful than the fitness level method.

The Peak problem in Definition 2.5 is to seek a needle in a haystack. The optimal solution is 1^n with fitness value 1, whereas all other solutions are with fitness value 0. The only possible $<_f$ -partition for the Peak problem contains two sets: \mathcal{S}_1 containing all non-optimal solutions and \mathcal{S}_2 containing the optimal solution. We study the running time of $(1+1)\text{-EA}^\neq$ on Peak. As introduced in Section 2.1, $(1+1)\text{-EA}^\neq$ is the same as $(1+1)\text{-EA}$ except that it does not accept solutions with equal fitness value in the selection step. Consequently, on the Peak problem, $(1+1)\text{-EA}$ performs a random walk in non-optimal solutions, as these solutions are with the same fitness value, whereas $(1+1)\text{-EA}^\neq$ stays until the optimal solution is found.

Theorem 5.2. \mathfrak{A}_{SA} is not reducible to \mathfrak{A}_{FL} .

The theorem is proved by comparing Lemmas 5.3 and 5.4. The proofs of these two lemmas are provided in Appendix A.2.

Lemma 5.3. For the process of (1+1)-EA $^{\neq}$ on the Peak problem, for all possible parameters, $\mathfrak{A}_{FL}^u \geq (1 - 1/2^n)n^n$ and $\mathfrak{A}_{FL}^l \leq (1 - 1/2^n)n(n/(n-1))^{n-1}$.

Lemma 5.4. For the process of (1+1)-EA $^{\neq}$ on the Peak problem, there exists an assignment of parameters such that $\mathfrak{A}_{SA}^u \leq (n/2 + n/(2(n-1)))^n$ and $\mathfrak{A}_{SA}^l \geq (n/2)^n$.

It is straightforward to verify by comparing Lemmas 5.3 and 5.4 that $\mathfrak{A}_{SA}^u < \mathfrak{A}_{FL}^u$ and $\mathfrak{A}_{SA}^l > \mathfrak{A}_{FL}^l$. Actually, the fitness level method can derive only a polynomial lower bound for this process, which is quite loose, whereas switch analysis can lead to tighter bounds as it allows a fine examination between fitness levels. In other words, the fitness level method cannot take all the input of switch analysis. Therefore, it is proved that \mathfrak{A}_{SA} is not reducible to \mathfrak{A}_{FL} .

5.3 Switch Analysis vs. Drift Analysis

In this section, we compare switch analysis with drift analysis. We first present the characterization of drift analysis, then prove that drift analysis is reducible to switch analysis, and finally examine a case study of the discrete linear problem, which shows that switch analysis can lead to new bounds of the running time.

Drift analysis [Hajek, 1982, Sasaki and Hajek, 1988, He and Yao, 2001, 2004] has been widely used to analyze the running time of EAs and has several variants [Happ et al., 2008, Oliveto and Witt, 2011, Doerr et al., 2012c, Doerr and Goldberg, 2013]. Here, we focus on the classical drift analysis approach in Theorem 2.3, which is also called additive drift analysis.

Intuitively, drift analysis introduces a distance function, not necessarily a metric, to measure the distance from any state to the target state space. Relying on the distance function, drift analysis estimates the average progress towards the target of every step of an EA, and then the number of steps the EA takes to arrive at the target can be derived through dividing the initial distance by one-step progress. When c_l in Theorem 2.3 is negative, drift analysis fails with such a distance function.

Note that the two recently proposed variants of drift analysis, i.e., multiplicative drift analysis in Theorem 2.4 and adaptive drift analysis [Doerr and Goldberg, 2013], are not stronger than the classical one, because multiplicative drift analysis is proved by the classical one and adaptive drift analysis is actually the classical (multiplicative) one with carefully-designed distance functions depending on the objective functions and some parameter values of the concerned EAs.

Characterization 5.3 (Drift Analysis). For an EA process $\xi \in \mathcal{X}$, the drift analysis approach \mathfrak{A}_{DA} is defined by:

Parameters: a distance function V .

Input:

$c_l > 0$ for upper bound analysis such that $\forall t \geq 0, \xi_t \notin \mathcal{X}^* : c_l \leq \mathbb{E}[V(\xi_t) - V(\xi_{t+1}) | \xi_t]$,

$c_u > 0$ for lower bound analysis such that $\forall t \geq 0, \xi_t \notin \mathcal{X}^* : c_u \geq \mathbb{E}[V(\xi_t) - V(\xi_{t+1}) | \xi_t]$.

Output:

$$\mathfrak{A}_{DA}^u = \sum_{x \in \mathcal{X}} \pi_0(x) V(x) / c_l,$$

$$\mathfrak{A}_{DA}^l = \sum_{x \in \mathcal{X}} \pi_0(x) V(x) / c_u.$$

Theorem 5.3. \mathfrak{A}_{DA} is reducible to \mathfrak{A}_{SA} .

The reducibility from drift analysis to switch analysis is proved by the following two lemmas. Lemma 5.5 shows the upper-bound reducibility. By using OneJump, whose parameters depend on the distance function V , as the reference chain and utilizing the input c_l of drift analysis, switch analysis can derive the same upper bound as drift analysis. The proof of Lemma 5.6 is similar to that of Lemma 5.5 by replacing c_l with c_u and changing the direction of the corresponding inequalities. These proofs are constructive, providing a way of using switch analysis to accomplish any results obtained by drift analysis.

Lemma 5.5. \mathfrak{A}_{DA} is upper-bound reducible to \mathfrak{A}_{SA} .

Proof. The proof is by constructing the parameters and input of \mathfrak{A}_{SA} from that of \mathfrak{A}_{DA} , such that $\mathfrak{A}_{SA}^u \leq \mathfrak{A}_{DA}^u$.

Denote ξ as the EA process to be analyzed. By \mathfrak{A}_{DA} in Characterization 5.3, we have a distance function V as the parameter, and c_l as the input. Let $\mathcal{V} = \{V(x) \mid x \in \mathcal{X}\} = \{V_0, V_1, \dots, V_m\}$ be the set of distinct values of the distance function, with the size of $m + 1$. We order the elements of \mathcal{V} as $V_0 = 0 < V_1 < \dots < V_m$, and denote the value index of x as $\mathcal{V}_x = i$ where $V(x) = V_i$. Without loss of generality, assume $V_1 \geq 1$, because if not, every element in \mathcal{V} as well as c_l can be multiplied with $1/V_1$, without affecting the drift condition and result.

We choose the reference process $\xi' \in \mathcal{Y} = \{0, 1\}^m$ to be the OneJump chain with dimension m and parameters $p_i = 1/V_{m-i} \forall 0 \leq i \leq m - 1$. We then construct the mapping function as $\forall x : \phi(x) = 1^{m-\mathcal{V}_x} 0^{\mathcal{V}_x}$, by using the distance function value index. This mapping function is optimal-aligned, because all populations with distance function value 0 are mapped to 1^m .

We examine Eq. (4.1). $\forall x \notin \mathcal{X}^*$, we have

$$\sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y]$$

$$\begin{aligned}
&= \sum_{i=0}^m P(V(\xi_{t+1}) = V_i \mid \xi_t = x) \cdot \mathbb{E}_{oj}(i) \\
&= \sum_{i=0}^m P(V(\xi_{t+1}) = V_i \mid \xi_t = x) \cdot V_i
\end{aligned} \tag{5.9}$$

$$= \mathbb{E}[V(\xi_{t+1}) \mid \xi_t = x], \tag{5.10}$$

where Eq. (5.9) holds by $\mathbb{E}_{oj}(i) = 1/p_{m-i}$. Because

$$\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] = V(x) - \mathbb{E}[V(\xi_{t+1}) \mid \xi_t = x] \geq c_l, \tag{5.11}$$

we have

$$\sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \leq V(x) - c_l. \tag{5.12}$$

Meanwhile, because $\phi(x) = 1^{m-\mathcal{V}_x} 0^{\mathcal{V}_x}$, we have, $\forall x \notin \mathcal{X}^*$,

$$\sum_{y \in \mathcal{Y}} P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] = \mathbb{E}_{oj}(\mathcal{V}_x) - 1 = V(x) - 1. \tag{5.13}$$

Combining Eqs. (5.12) and (5.13), we have, $\forall x \notin \mathcal{X}^*$,

$$\begin{aligned}
&\sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\
&- \sum_{y \in \mathcal{Y}} P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] \leq 1 - c_l.
\end{aligned} \tag{5.14}$$

By summing up all x , we have

$$\begin{aligned}
&\sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_t(x) P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\
&\leq \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_t(x) P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] + (1 - c_l) \cdot (1 - \pi_t(\mathcal{X}^*)).
\end{aligned} \tag{5.15}$$

Therefore, $\rho_t^u = (1 - c_l) \cdot (1 - \pi_t(\mathcal{X}^*))$ is a proper assignment of ρ_t^u in Characterization 5.1. We then calculate

$$\rho^u = \sum_{t=0}^{+\infty} \rho_t^u = (1 - c_l) \cdot \sum_{t=0}^{+\infty} (1 - \pi_t(\mathcal{X}^*)) = (1 - c_l) \cdot \mathbb{E}[\tau \mid \xi_0 \sim \pi_0], \tag{5.16}$$

and thus,

$$\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \leq \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] + (1 - c_l) \cdot \mathbb{E}[\tau \mid \xi_0 \sim \pi_0], \tag{5.17}$$

implying $\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \leq \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi]/c_l$. By the definition of the mapping function ϕ , we have

$$\begin{aligned}\mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] &= \sum_{i=0}^m \pi_0(\{\mathbf{x} \mid \mathcal{V}_{\mathbf{x}} = i\}) \cdot \mathbb{E}_{oj}(i) \\ &= \sum_{i=0}^m \pi_0(\{\mathbf{x} \mid V(\mathbf{x}) = V_i\}) \cdot V_i = \sum_{\mathbf{x} \in \mathcal{X}} \pi_0(\mathbf{x}) \cdot V(\mathbf{x}).\end{aligned}\quad (5.18)$$

Thus, we have

$$\mathfrak{A}_{SA}^u = \frac{1}{c_l} \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] = \frac{1}{c_l} \sum_{\mathbf{x} \in \mathcal{X}} \pi_0(\mathbf{x}) \cdot V(\mathbf{x}) = \mathfrak{A}_{DA}^u,\quad (5.19)$$

which proves the lemma. \square

Lemma 5.6. \mathfrak{A}_{DA} is lower-bound reducible to \mathfrak{A}_{SA} .

We are then interested in examining whether switch analysis is reducible to drift analysis. It is, however, hard to obtain a full answer, because that would require to examine all possible distance functions in an unconstrained function space. By studying the discrete linear problem, however, we show that switch analysis is not reducible to a restricted version of drift analysis that uses any fixed linear distance function.

Definition 5.4 (Discrete Linear Problem). A discrete linear problem with size n and vocabulary set $\{0, 1, \dots, r\}$ is to find a solution $\mathbf{s}^* \in \{0, 1, \dots, r\}^n$ such that, for given positive weights w_1, w_2, \dots, w_n ,

$$\mathbf{s}^* = \arg \max_{\mathbf{s} \in \{0, 1, \dots, r\}^n} \sum_{i=1}^n w_i s_i,\quad (5.20)$$

where s_i denotes the i -th element of a solution $\mathbf{s} \in \{0, 1, \dots, r\}^n$. Without loss of generality, assume that $w_1 \leq w_2 \leq \dots \leq w_n$.

To run (1+1)-EA on the discrete linear problem, the mutation operator needs to be modified. When the input space is $\{0, 1\}^n$, the mutation operator flips a bit of a solution from 0 to 1 and vice versa. While the input space is $\{0, 1, \dots, r\}^n$, the mutation operator is modified to flip a bit from its own value k to a random value in $\{0, 1, \dots, r\} \setminus \{k\}$. It has been proved that, for the discrete linear problem in Definition 5.4, there exists no universal linear distance function such that (1+1)-EA has a positive drift [Doerr et al., 2011b, 2012b], and thus \mathfrak{A}_{DA}^u fails.

Definition 5.5 (Linear Distance Function Space). For solutions of length n , the linear distance function space \mathcal{L} consists of all distance functions that are linear combination of solution bits, i.e.,

$$\mathcal{L} = \left\{ V \mid \forall \mathbf{w} \in \mathbb{R}^n : V(\mathbf{s}) = \sum_{i=1}^n w_i s_i \right\}.\quad (5.21)$$

Lemma 5.7. [Doerr et al., 2011b, 2012b] For the process of

- (1) (1+1)-EA with any mutation probability $p \geq 7/n$ on the discrete linear problem with vocabulary $\{0, 1\}$;
- (2) (1+1)-EA with the mutation probability $p = 1/n$ on the discrete linear problem with vocabulary $\{0, 1, \dots, r\}$ where $r \geq 43$,
 \mathfrak{A}_{DA}^u with any fixed parameter $V \in \mathcal{L}$ fails.

Theorem 5.4. For the process of (1+1)-EA with the mutation probability $p = c/n$ for some constant $c > 0$ on the discrete linear problem,

- (1) with vocabulary $\{0, 1\}$, there exists an assignment of parameters such that $\mathfrak{A}_{SA}^u = O(n \log n)$;
- (2) with vocabulary $\{0, 1, \dots, r\}$, there exists an assignment of parameters such that $\mathfrak{A}_{SA}^u = (1 + o(1))(e^c/c)rn \log n + O(r^3n \log \log n)$.

Theorem 5.4 can be proved by applying Theorem 5.3, i.e., drift analysis is reducible to switch analysis, and considering the bounds proved in [Doerr and Pohl, 2012] and [Doerr and Goldberg, 2013] using multiplicative adaptive drift analysis. Comparing Theorem 5.4 with Lemma 5.7, switch analysis is not reducible to the restricted version of drift analysis.

The bound in Theorem 5.4(b) involves r^3 . In Theorem 5.5 we give another upper bound using switch analysis, which is tighter in terms of r although looser in terms of n . The reference process used in the proof of Theorem 5.5 is RLS $^\neq$ running on the LeadingOnes problem. As introduced in Section 2.1, RLS $^\neq$ is a modification of RLS, where the only difference lies in the fact that RLS $^\neq$ uses the strict selection strategy. In other words, RLS accepts the offspring solution with equal fitness, whereas RLS $^\neq$ accepts only a better offspring solution. We denote the reference process as ξ' , and thus the first hitting time as τ' , which has a property in Lemma 5.8. The notation is simplified by denoting $\mathbb{E}_{rls}(j)$ as the CFHT $\mathbb{E}[\tau' \mid \xi'_t = s]$ with $|s|_0 = j$, i.e., $\mathbb{E}_{rls}(j) = n \cdot j$. The proofs of Lemma 5.8 and Theorem 5.5 are provided in Appendix A.2.

Lemma 5.8. $\forall t \geq 0, \forall s \in \{0, 1\}^n : \mathbb{E}[\tau' \mid \xi'_t = s] = n \cdot |s|_0$.

Theorem 5.5. For the process of (1+1)-EA with any mutation probability $p \in (0, 0.5)$ on the discrete linear problem with vocabulary $\{0, 1, \dots, r\}$, there exists an assignment of parameters such that $\mathfrak{A}_{SA}^u \leq r^2n/(p(1-p)^{n-1})$.

The upper bound $r^2n/(p(1-p)^{n-1})$ arrives at its minimum of $O(r^2n^2)$ at $p = 1/n$. Now we know that the expected running time of (1+1)-EA with the mutation rate $1/n$, on the discrete linear problem with vocabulary $\{0, 1, \dots, r\}$, is $O(\min\{r^2n^2, rn \log n + r^3n \log \log n\})$.

5.4 Switch Analysis vs. Convergence-based Analysis

In this section, we compare switch analysis with convergence-based analysis. We first present the characterization of convergence-based analysis,

then prove that convergence-based analysis is reducible to switch analysis, and finally examine a case study of the constrained Trap problem, showing that switch analysis can lead to tighter bounds than that obtained by convergence-based analysis.

The convergence-based analysis approach as presented in Theorem 3.1 requires to estimate the probabilities for the algorithm to achieve the optimal solution in every step. The expected running time is then calculated based on these probabilities. To study reducibility between convergence-based analysis and switch analysis, we present its characterization in Characterization 5.4. Note that the convergence-based analysis approach does not have parameters.

Characterization 5.4 (Convergence-based Analysis). *For an EA process $\xi \in \mathcal{X}$, the convergence-based analysis approach \mathfrak{A}_{CA} is defined by:*

Input:

$$\alpha_t \leq \sum_{x \notin \mathcal{X}^*} P(\xi_{t+1} \in \mathcal{X}^* \mid \xi_t = x) \frac{\pi_t(x)}{1 - \pi_t(\mathcal{X}^*)} \quad \forall t \geq 0, \text{ with a restriction that}$$

$$\prod_{t=0}^{+\infty} (1 - \alpha_t) = 0,$$

$$\beta_t \geq \sum_{x \notin \mathcal{X}^*} P(\xi_{t+1} \in \mathcal{X}^* \mid \xi_t = x) \frac{\pi_t(x)}{1 - \pi_t(\mathcal{X}^*)} \quad \forall t \geq 0.$$

Output:

$$\mathfrak{A}_{CA}^u = (1 - \pi_0(\mathcal{X}^*)) (\sum_{t=1}^{+\infty} t \alpha_{t-1} \prod_{i=0}^{t-2} (1 - \alpha_i)),$$

$$\mathfrak{A}_{CA}^l = (1 - \pi_0(\mathcal{X}^*)) (\sum_{t=1}^{+\infty} t \beta_{t-1} \prod_{i=0}^{t-2} (1 - \beta_i)).$$

Theorem 5.6. \mathfrak{A}_{CA} is reducible to \mathfrak{A}_{SA} .

Before proving the theorem, we introduce a simple Markov chain called OneJump-fix, which will be used as the reference chain in switch analysis. For OneJump-fix, a non-target state either jumps to a target state or stays as it is in one-step with a fixed probability.

Definition 5.6 (OneJump-fix). *The OneJump-fix chain is a homogeneous Markov chain $\xi \in \mathcal{X}$ with a parameter $p_{fix} \in [0, 1]$ and target subspace \mathcal{X}^* . Its transition probability is defined as, $\forall t \geq 0, x \in \mathcal{X}$:*

$$P(\xi_{t+1} = y \mid \xi_t = x) = \begin{cases} p_{fix}, & y \in \mathcal{X}^*; \\ 1 - p_{fix}, & y = x; \\ 0, & \text{otherwise.} \end{cases} \quad (5.22)$$

Theorem 5.6 holds by combining Lemmas 5.9 and 5.10, which prove the upper-bound and lower-bound reducibility, respectively.

Lemma 5.9. \mathfrak{A}_{CA} is upper-bound reducible to \mathfrak{A}_{SA} .

Proof. The proof is by finding the parameters of \mathfrak{A}_{SA} and the input of \mathfrak{A}_{SA} from that of \mathfrak{A}_{CA} , and showing that $\mathfrak{A}_{SA}^u \leq \mathfrak{A}_{CA}^u$.

Denote $\xi \in \mathcal{X}$ as the EA process to be analyzed. We know the variable α_t in Characterization 5.4 as the input of \mathfrak{A}_{CA} . We choose the reference chain

ξ' as the OneJump-fix chain in the same space of ξ , i.e., the state space \mathcal{X} and the target subspace \mathcal{X}^* . The parameter of ξ' is set as

$$p_{fix} = \frac{1}{\sum_{t=0}^{+\infty} \prod_{i=0}^{t-1} (1 - \alpha_i)}, \quad (5.23)$$

implying $\forall t \geq 0, x \notin \mathcal{X}^* : \mathbb{E}[\tau' | \xi'_t = x] = \sum_{t=0}^{+\infty} \prod_{i=0}^{t-1} (1 - \alpha_i)$. We design the mapping function $\phi : \mathcal{X} \rightarrow \mathcal{Y}$ as $\forall x : \phi(x) = x$, which is optimal-aligned.

Next we calculate the upper bound output of \mathfrak{A}_{SA} using the input of \mathfrak{A}_{CA} and the reference process. For the left part of Eq. (4.1), we have

$$\begin{aligned} & \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_t(x) P(\xi_{t+1} \in \phi^{-1}(y) | \xi_t = x) \mathbb{E}[\tau' | \xi'_0 = y] \\ &= \sum_{x \in \mathcal{X}} \pi_t(x) (1 - P(\xi_{t+1} \in \mathcal{X}^* | \xi_t = x)) \sum_{t=0}^{+\infty} \prod_{i=0}^{t-1} (1 - \alpha_i) \\ &= (1 - \pi_{t+1}(\mathcal{X}^*)) \sum_{t=0}^{+\infty} \prod_{i=0}^{t-1} (1 - \alpha_i). \end{aligned} \quad (5.24)$$

For the right part of Eq. (4.1), we have

$$\begin{aligned} & \sum_{u, y \in \mathcal{Y}} \pi_t^\phi(u) P(\xi'_1 = y | \xi'_0 = u) \mathbb{E}[\tau' | \xi'_1 = y] \\ &= \sum_{x \in \mathcal{X} \setminus \mathcal{X}^*} \pi_t(x) \left(\sum_{t=0}^{+\infty} \prod_{i=0}^{t-1} (1 - \alpha_i) - 1 \right) \end{aligned} \quad (5.25)$$

$$= (1 - \pi_t(\mathcal{X}^*)) \left(\sum_{t=0}^{+\infty} \prod_{i=0}^{t-1} (1 - \alpha_i) - 1 \right), \quad (5.26)$$

where Eq. (5.25) holds by Lemma 2.1. Thus, $\forall t \geq 0$, we have

$$\begin{aligned} & \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_t(x) P(\xi_{t+1} \in \phi^{-1}(y) | \xi_t = x) \mathbb{E}[\tau' | \xi'_0 = y] \\ & - \sum_{u, y \in \mathcal{Y}} \pi_t^\phi(u) P(\xi'_1 = y | \xi'_0 = u) \mathbb{E}[\tau' | \xi'_1 = y] \\ &= (\pi_t(\mathcal{X}^*) - \pi_{t+1}(\mathcal{X}^*)) \sum_{t=0}^{+\infty} \prod_{i=0}^{t-1} (1 - \alpha_i) + (1 - \pi_t(\mathcal{X}^*)) \end{aligned} \quad (5.27)$$

$$\leq (\pi_t(\mathcal{X}^*) - \pi_{t+1}(\mathcal{X}^*)) \sum_{t=0}^{+\infty} \prod_{i=0}^{t-1} (1 - \alpha_i) + (1 - \pi_0(\mathcal{X}^*)) \prod_{i=0}^{t-1} (1 - \alpha_i), \quad (5.28)$$

where Eq. (5.27) holds by combining Eqs. (5.24) and (5.26), and Eq. (5.28) holds by Lemma 3.1. Therefore, a proper assignment of ρ_t^u is found, and

$$\begin{aligned} \rho^u &= \sum_{t=0}^{+\infty} (\pi_t(\mathcal{X}^*) - \pi_{t+1}(\mathcal{X}^*)) \sum_{t=0}^{+\infty} \prod_{i=0}^{t-1} (1 - \alpha_i) + \sum_{t=0}^{+\infty} (1 - \pi_0(\mathcal{X}^*)) \prod_{i=0}^{t-1} (1 - \alpha_i) \\ &= \left(\pi_0(\mathcal{X}^*) - \lim_{t \rightarrow +\infty} \pi_t(\mathcal{X}^*) + 1 - \pi_0(\mathcal{X}^*) \right) \sum_{t=0}^{+\infty} \prod_{i=0}^{t-1} (1 - \alpha_i) = 0, \quad (5.29) \end{aligned}$$

where Eq. (5.29) holds because the chain converges to \mathcal{X}^* by Lemma 3.1. Considering $\pi_0^\phi(y) = \pi_0(y)$, we have

$$\begin{aligned} \mathfrak{A}_{SA}^u &= \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] + 0 \\ &= (1 - \pi_0(\mathcal{X}^*)) \sum_{t=0}^{+\infty} \prod_{i=0}^{t-1} (1 - \alpha_i) \\ &= (1 - \pi_0(\mathcal{X}^*)) \left(1 + \sum_{t=2}^{+\infty} t \left(\prod_{i=0}^{t-2} (1 - \alpha_i) - \prod_{i=0}^{t-1} (1 - \alpha_i) \right) - (1 - \alpha_0) \right) \\ &= (1 - \pi_0(\mathcal{X}^*)) \left(\sum_{t=1}^{+\infty} t \alpha_{t-1} \prod_{i=0}^{t-2} (1 - \alpha_i) \right) = \mathfrak{A}_{CA}^u, \quad (5.30) \end{aligned}$$

which proves the lemma. \square

Lemma 5.10. \mathfrak{A}_{CA} is lower-bound reducible to \mathfrak{A}_{SA} .

Proof. The proof is similar to that of Lemma 5.9 except the following minor changes: “ α ” and “ \leq ” are replaced by “ β ” and “ \geq ”, respectively; the last “=” in Eq. (5.29) and the first “=” in Eq. (5.30) are replaced by “ \geq ” because the convergence cannot be derived using only the input β . \square

Next, we compare convergence-based analysis with switch analysis in a case study of solving the constrained Trap problem by (1+1)-EA or RLS. We first introduce a simple Markov chain, TrapJump, which will be used as the reference chain for switch analysis. The TrapJump chain is a simple Markov chain with only $(n + 1)$ states, named 0 to n . For non-optimal state $i \in [n]$, there are only three possible one-step transitions: jumping to the optimal state 0, moving one step farther from the optimal state, i.e., moving to state $i + 1$, or staying where it is, i.e., staying at state i . The farther from the optimal state, the lower the probability for transiting to the optimal state in one step. This chain reflects our intuition on the deceptiveness of the constrained Trap problem.

Definition 5.7 (TrapJump). The TrapJump chain with state space dimension n is a homogeneous Markov chain $\xi \in \{0, 1, \dots, n\}$ with a parameter $p \in (0, 0.5)$ and target state 0. Its transition probability is defined as, $\forall t \geq 0$,

$$\forall i > 1 : P(\xi_{t+1} = y \mid \xi_t = i) = \begin{cases} p^i(1-p)^{n-i}, & y = 0; \\ (n-i)p(1-p)^{n-1}, & y = i+1; \\ 1-p^i(1-p)^{n-i}, & y = i; \\ -(n-i)p(1-p)^{n-1}, & otherwise, \end{cases} \quad (5.31)$$

$$and \quad P(\xi_{t+1} = y \mid \xi_t = 1) = \begin{cases} p(1-p)^{n-1}, & y = 0; \\ p(1-p)^{n-1}, & y = 2; \\ 1-2p(1-p)^{n-1}, & y = 1; \\ 0, & otherwise. \end{cases} \quad (5.32)$$

Let $\mathbb{E}_{tj}(i)$ denote the CFHT of TrapJump starting from state i . $\mathbb{E}_{tj}(0) = 0$ implies the optimal state, and we have the following lemma.

Lemma 5.11. $\forall i \geq 1 : \mathbb{E}_{tj}(i) \geq 1/(p^i(1-p)^{n-i})$, and $\mathbb{E}_{tj}(i-1) \leq \mathbb{E}_{tj}(i)$.

Theorem 5.7. *For the process of (1+1)-EA on the constrained Trap problem, there exists an assignment of parameters such that $\mathfrak{A}_{SA}^l = \Omega(1/(2p(1-p))^n)$, where $p \in (0, 0.5)$ is the mutation probability.*

Theorem 5.8. *For the process of RLS on the constrained Trap problem, there exists an assignment of parameters such that $\mathfrak{A}_{SA}^l = \Omega((n/2)^n)$.*

The proofs are provided in Appendix A.2. In Chapter 3, convergence-based analysis has been used to derive the expected running time of (1+1)-EA and RLS on constrained Trap, and the results are: $\mathfrak{A}_{CA}^l = \Omega(1/(p(1-p)^n))$ for (1+1)-EA, and $\mathfrak{A}_{CA}^l = \Omega(2^n)$ for RLS. Theorems 5.7 and 5.8 show that $\mathfrak{A}_{SA}^l = \Omega(1/(2p(1-p))^n)$ for (1+1)-EA and $\mathfrak{A}_{SA}^l = \Omega((n/2)^n)$ for RLS. In both cases, switch analysis achieves tighter lower bounds.

5.5 Analysis Approaches: Unification

Now, we have shown that fitness level, drift analysis, and convergence-based analysis are all reducible to switch analysis. As the proofs of reducibility are constructive, we can find configurations of switch analysis, equivalent to these three approaches. By comparing these configurations, we have an opportunity to compare these approaches in a unified framework.

The proofs of reducibility of both fitness level and drift analysis to switch analysis, i.e., Theorems 5.1 and 5.3, use the OneJump chain in Definition 5.3 as the reference chain, whereas the proof of reducibility of convergence-based analysis to switch analysis, i.e., Theorem 5.6, uses the OneJump-fix chain in Definition 5.6 as the reference chain. OneJump is similar to OneJump-fix, as both of them are simply jumping to a target state or staying where they are. For OneJump, the transition probability depends on the solution, whereas the transition probability for OneJump-fix is a constant. In

addition to the transition probability, they are different in the state space. The OneJump chain is in a binary state space, whereas the OneJump-fix chain is in the same state space with the concerned evolutionary process.

For fitness level, the solution space is partitioned into m subspaces $\{\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_m\}$, and the state space of the reference chain is set to $\{0, 1\}^{m-1}$. A state belonging to \mathcal{S}_i is mapped to the binary solution $1^{i-1}0^{m-i}$. For drift analysis, \mathcal{V} denotes the set of all distinct values of the distance function V , of which the size is $m + 1$, and the state space of the reference chain is set to $\{0, 1\}^m$. Moreover, \mathcal{V}_x is used to denote the ordered index of the distance value for the state x . A state x is mapped to $1^{m-\mathcal{V}_x}0^{\mathcal{V}_x}$. For convergence-based analysis, the reference chain has the same state space with the concerned evolutionary process, and a state is simply mapped to itself.

Table 5.1 summarizes the major components of the reduction of the three analysis approaches to switch analysis, allowing us to make comparisons. We can observe a limitation of fitness level: the aligned mapping must make some different solutions collapse into one binary string. This may make the method unable to distinguish some solutions that need to be handled differently. The limitation of convergence-based analysis is in the other end: the transition probability is the same for all solutions, and thus cannot be configured to handle solutions differently. This may make the approach less flexible. For drift analysis, when the distance function can have a different value on every solution, it can distinguish the solutions by the aligned mapping, and meanwhile, assign different transition probabilities to every solution. It also reminds us that choosing a suitable distance function is vital for drift analysis, though this is not straightforward.

Table 5.1. Comparison of the analysis approaches, i.e., fitness level, drift analysis and convergence-based analysis, from the reduction to switch analysis.

Approach	Reference chain	Transition probability	Aligned mapping
Fitness level	OneJump in $\{0, 1\}^{m-1}$ with m being the number of levels	$p_i = \frac{1}{\frac{1}{v_{i+1}} + x_u \sum_{j=i+2}^{m-1} \frac{1}{v_j}}$	$\phi(x) = 1^{i-1}0^{m-i}$
Drift analysis	OneJump in $\{0, 1\}^m$ with $(m + 1)$ being the number of distinct distance values	$p_i = \frac{1}{\sqrt{m-i}}$	$\phi(x) = 1^{m-\mathcal{V}_x}0^{\mathcal{V}_x}$
Convergence-based analysis	OneJump-fix in \mathcal{X} , i.e., the original state space	$p_{fix} = \frac{1}{\sum_{t=0}^{+\infty} \prod_{i=0}^{t-1} (1-\alpha_i)}$	$\phi(x) = x$

The above discussion on the reducibility relationship among different approaches, however, is only about some theoretical ability of these approaches, rather than their “goodness” or other aspects. For example, the hardness of using an analysis approach may depend on the background knowledge, problem understanding, and personal preference of an analyst.

5.6 Summary

To study the relative strength between general analysis approaches for EAs, this chapter formally characterizes these approaches and defines the concept of reducibility. The reducibility is defined by following the intuition that an approach is at least as powerful as another if results derived by it can be no worse without using more information. We have shown that the fitness level method, drift analysis, and convergence-based analysis are all reducible to switch analysis. On the opposite direction, we have shown that switch analysis is not reducible to fitness level by studying the Peak problem, and switch analysis is not reducible to a restricted version of drift analysis by studying the discrete linear problem. We have also shown that switch analysis can derive tighter bounds than convergence-based analysis by studying the constrained Trap problem. These results disclose the power of switch analysis for running time analysis of EAs. Moreover, from the constructive proofs of reducibility, we can observe that fitness level, drift analysis or convergence-based analysis is equivalent to some configuration of switch analysis. Through comparing the equivalent configurations, we discuss the differences among these three approaches.

If one has already obtained a running time bound using the fitness level method, drift analysis or convergence-based analysis, there can be a simple way to use switch analysis to improve the bound further. Note that the proofs of the reducibility in Theorems 5.1, 5.3 and 5.6 are constructive. One can first transform the analysis process to that by switch analysis, and then try to replace some components of switch analysis, such as the reference process or the mapping function, to improve the bound.



6

Approximation Analysis: SEIP

When EAs are applied to tackle hard problems, particularly some NP-hard problems, e.g., [Higuchi et al., 1999, Koza et al., 2003, Hornby et al., 2006, Benkhelifa et al., 2009], they are often expected to obtain *good enough* solutions, rather than optimal solutions. This chapter studies a largely under-explored issue: the approximation performance of EAs, i.e., how close the obtained solution is to an optimal solution.

Previously, He and Yao [2003b] studied conditions under which the *wide-gap far distance* and *narrow-gap long distance* problems are hard for EAs to approximate. Giel and Wegener [2003] studied (1+1)-EA on the *maximum matching* problem, and found that it takes exponential time to find optimal solutions whereas only $O(n^{2\lceil 1/\epsilon \rceil})$ time for $(1/(1+\epsilon))$ -approximate solutions, demonstrating the value of using EAs for approximation.

Later on, more results of applying EAs for approximate solutions have been derived. For (1+1)-EA which is the simplest version of EAs, the results are in two folds. On the one hand, it has been found that (1+1)-EA has an arbitrarily bad approximation ratio on the *vertex cover* problem and the *set cover* problem [Oliveto et al., 2009a, Friedrich et al., 2010]. On the other hand, it has been found that, (1+1)-EA is of the *polynomial-time randomized approximation scheme* on a subclass of the *partition* problem [Witt, 2005], and on some instances of the vertex cover problem where (1+1)-EA gets stuck, multiple restarts strategy can let the algorithm have a large probability to find optimal solutions [Oliveto et al., 2009a]. Another result for (1+1)-EA shows that it improves a 2-approximation algorithm to $(2 - 2/n)$ on the vertex cover problem [Friedrich et al., 2007], implying that it may be useful as a post-optimizer.

Recent advances in MOEAs shed light on the power of EAs as approximation optimizers. For a single-objective optimization problem, the *multi-objective reformulation* introduces an auxiliary objective function, and then an MOEA is employed as the optimizer. Scharnow et al. [2002] first indicated the potential that the multi-objective reformulation can be superior to tackling the problem directly using a single-objective EA. This has been

confirmed on various problems [Neumann and Wegener, 2006, Neumann and Reichel, 2008, Friedrich et al., 2010, Neumann et al., 2011] by showing that MOEAs using auxiliary objective functions can solve some problems efficiently, whereas single-objective EAs may get stuck.

As for approximation, MOEAs have been shown effective on some NP-hard problems: in [Neumann and Reichel, 2008], MOEAs have been proved to achieve a k -approximation ratio on the *minimum multicuts* problem in polynomial time; in [Friedrich et al., 2010], an MOEA has been proved to achieve an H_m -approximation ratio on the set cover problem in polynomial time, reaching the asymptotic lower bound, where m is the size of the ground set.

In this chapter, we present a general framework [Yu et al., 2012] for studying the approximation ability of EAs. The presented framework called Simple EAs with Isolated Population (SEIP) adopts an *isolation function* to manage the competition among solutions. By specifying the isolation function, SEIP can be implemented as single-objective EAs as well as MOEAs. The MOEAs analyzed in [Neumann and Laumanns, 2006, Neumann and Reichel, 2008, Friedrich et al., 2010] can be viewed as special cases of SEIP, in term of the solutions maintained in the population. By analyzing the SEIP framework, we get a general characterization of EAs for guaranteed approximation solutions. As an application illustration, we study set cover, an NP-hard problem. We prove that a simple configuration of SEIP efficiently obtains an H_m -approximation ratio,[†] the asymptotic lower bound for the set cover problem [Feige, 1998].

The rest of this chapter is organized as follows. Section 6.1 presents the framework of SEIP and characterizes its general approximation behaviors. The approximation ratios of SEIP are then analyzed on the set cover problem in Section 6.2. Section 6.3 concludes this chapter.

6.1 SEIP: Framework

Before introducing the SEIP framework, we first introduce some preliminaries on minimization problems. The minimization problem is generally defined as follows.

Definition 6.1 (Minimization Problem). *Given an evaluation function f and a set \mathcal{C} of feasibility constraints, to find a solution $s \in \{0, 1\}^n$ which minimizes $f(s)$ while satisfying constraints in \mathcal{C} . A problem instance can be specified by its parameters (n, f, \mathcal{C}) .*

In the above definition, solutions are represented in binary vectors. When a minimization problem aims at finding a subset from a universe

[†]More specifically, the approximation ratio is the Harmonic number of the cardinality of the largest set.

set, a binary vector can be equivalently used to represent a subset, as introduced in Section 1.2. Each element of a vector corresponds to a member of the universe set. For example, given a universe set $\{1, 2, 3, 4, 5\}$, its subset $\{1, 3, 5\}$ can be represented by a binary vector $(1, 0, 1, 0, 1)$. Considering the equivalence between sets and binary vectors, set operators (\cap , \cup , \setminus , \subseteq , \in) will be applied to binary vectors when there is no confusion. For example, $(1, 0, 1, 0, 1) \cap (0, 0, 0, 1, 1) = (0, 0, 0, 0, 1)$, $(1, 0, 1, 0, 1) \cup (0, 0, 0, 1, 1) = (1, 0, 1, 1, 1)$ and $(1, 0, 1, 0, 1) \setminus (0, 0, 0, 1, 1) = (1, 0, 1, 0, 0)$. Denote $s^\emptyset = (0, 0, \dots, 0)$ as the vector corresponding to the empty set.

As introduced in Section 3.2, a solution is feasible if it satisfies all constraints in \mathcal{C} . Otherwise, it is an infeasible solution, or a *partial* solution. Assume that the evaluation function f is defined on the full input space, i.e., it evaluates all possible solutions regardless of their feasibility.

Given a minimization problem, let s^{OPT} denote an optimal solution of the problem, i.e., $f(s^{\text{OPT}}) = \text{OPT}$. For a feasible solution s , we call the ratio $f(s)/\text{OPT}$ as its *approximation ratio*. If the approximation ratio of a feasible solution is upper bounded by some value r , i.e., $1 \leq f(s)/\text{OPT} \leq r$, the solution is called an *r -approximate solution*. The approximation ratio has been defined for maximization in Section 2.3, whereas minimization is considered here. An algorithm that guarantees to find an r -approximate solution for an arbitrary problem instance in polynomial time is an *r -approximation algorithm*.

The multi-objective reformulation, which employs an MOEA, such as SEMO/GSEMO in Algorithm 2.5 and DEMO [Neumann and Reichel, 2008], to tackle a single-objective optimization problem, has been studied and shown to achieve good approximation ratios on some problems. We hypothesize that the approximation power of the multi-objective reformulation owes to the fact that the non-dominated relationship in MOEAs isolates competitions between solutions; that is, if two solutions are non-dominated by each other, they do not compete for survival. We implement a direct isolated population strategy in Algorithm 6.1.

The mutation operator can use either one-bit mutation or bit-wise mutation. Usually, one-bit mutation is considered as local search, whereas bit-wise mutation relates to global search. We name SEIP using one-bit mutation as LSEIP, and that using bit-wise mutation as GSEIP, where letters “L” and “G” denote “local” and “global”, respectively.

SEIP employs an isolation function μ to isolate solutions. The isolation function maps a solution in $\{0, 1\}^n$ to a subset of $[q]$ for some integer $q \in \mathbb{N}^+$. Two solutions s and s' compete with each other only when $|\mu(s)| = |\mu(s')|$. In that case, we say the two solutions are in the same isolation.

Compared with single-objective EAs where competitions are between the offspring solutions and all solutions in the current population, SEIP employs the isolation function such that an offspring solution competes only with its parent solutions in the same isolation. When the isolation function

Algorithm 6.1 SEIP Framework

Input: minimization problem (n, f, \mathcal{C}) ; isolation function $\mu : \{0, 1\}^n \rightarrow 2^{[q]}$ for some integer $q \in \mathbb{N}^+$

Output: set of solutions from $\{0, 1\}^n$

Process:

```

1: let  $P = \{\mathbf{s}^\emptyset = (0, 0, \dots, 0)\}$ ;
2: while criterion is not met do
3:   select a solution  $\mathbf{s}$  from  $P$  uniformly at random;
4:   apply mutation on  $\mathbf{s}$  to generate  $\mathbf{s}'$ ;
5:   if  $\forall \mathbf{s} \in P : \text{superior}(\mathbf{s}, \mathbf{s}') = \text{false}$  then
6:      $Q = \{\mathbf{s} \in P \mid \text{superior}(\mathbf{s}', \mathbf{s}) = \text{true}\}$ ;
7:      $P = (P \setminus Q) \cup \{\mathbf{s}'\}$ 
8:   end if
9: end while
10: return  $P$ 

```

where the `superior` function determines whether a solution is superior to the other. Formally, $\text{superior}(\mathbf{s}, \mathbf{s}')$ is true if both (1) and (2) are satisfied:

- (1) $|\mu(\mathbf{s})| = |\mu(\mathbf{s}')|$;
- (2) $f(\mathbf{s}) < f(\mathbf{s}')$, or $f(\mathbf{s}) = f(\mathbf{s}')$ but $|\mathbf{s}|_1 < |\mathbf{s}'|_1$,

and $\text{superior}(\mathbf{s}, \mathbf{s}')$ is false otherwise.

puts all solutions in an isolation, SEIP degrades to an algorithm similar to (1+1)-EA in Algorithm 2.1.

The isolation function can also be configured to simulate the domination relationship of MOEAs such as SEMO/GSEMO in Algorithm 2.5 and DEMO [Neumann and Reichel, 2008]. If we are dealing with m -objective optimization with discrete objective values, a simple approach is to use one of the objective functions, say f_1 , as the fitness function and use the combination of the remaining $(m - 1)$ objective functions, say f_2, \dots, f_m , as the isolation function. Thus, two solutions compete for f_1 only when they share the same objective values for f_2, \dots, f_m . This simulation shows that all non-dominated solutions of an MOEA are kept in the population of SEIP, and if a solution does not reside in the population of SEIP, there must exist another solution dominating it. Hence, SEIP can be viewed as a generalization of MOEAs in term of the solutions retained. This simulation also reveals that SEIP can retain more solutions than an MOEA using the domination relationship and hence has more opportunities to find a better approximation solution. Though SEIP takes more time to manipulate a larger population than an MOEA does, the time cost can be reduced by defining an isolation function which aggregates nearby solutions, as has been done for DEMO [Neumann and Reichel, 2008].

The stop criterion is not described in Algorithm 6.1, because EAs are usually used as anytime algorithms in practice. We will analyze the approximation ratio and the corresponding computational complexity of SEIP.

For minimization problems, the isolation function μ is required to be *linearly additive*; that is,

$$\begin{cases} \mu(s) = [q], & \text{for all feasible solutions } s; \\ \mu(s \cup s') = \mu(s) \cup \mu(s'), & \text{for all solutions } s \text{ and } s'. \end{cases} \quad (6.1)$$

The quality of feasible solutions is measured by the approximation ratio. For partial (infeasible) solutions, we need a similar measure to evaluate how good the partial solution is. For this purpose, we define the *partial reference function* and *partial ratio* as follows.

Definition 6.2 (Partial Reference Function). *Given a minimization problem (n, f, \mathcal{C}) and an isolation function μ , a function $\mathcal{L} : 2^{[q]} \rightarrow \mathbb{R}$ is a partial reference function with respect to μ , if*

- (1) $\mathcal{L}([q]) = \text{OPT};$
- (2) $\mathcal{L}(R_1) = \mathcal{L}(R_2) \forall R_1, R_2 \subseteq [q] \text{ such that } |R_1| = |R_2|.$

Definition 6.3 (Partial Ratio). *Given a minimization problem (n, f, \mathcal{C}) , an isolation function μ and its partial reference function \mathcal{L} , the partial ratio of a (partial) solution s is*

$$\frac{f(s)}{\mathcal{L}(\mu(s))}; \quad (6.2)$$

the conditional partial ratio of s' conditioned on s is

$$\frac{f(s' | s)}{\mathcal{L}(\mu(s') | \mu(s))}, \quad (6.3)$$

where $f(s'|s) = f(s \cup s') - f(s)$ and $\mathcal{L}(\mu(s') | \mu(s)) = \mathcal{L}(\mu(s) \cup \mu(s')) - \mathcal{L}(\mu(s))$.

The partial ratio is a natural extension of the approximation ratio. Note that for a feasible solution, its partial ratio is equal to its approximation ratio. We derive two properties of the partial ratio. One is that the partial ratio is non-increasing in SEIP as stated in Lemma 6.1, and the other is its decomposability as stated in Lemma 6.2.

Lemma 6.1. *Given a minimization problem (n, f, \mathcal{C}) , an isolation function μ and its partial reference function \mathcal{L} , if SEIP has generated an offspring solution s with partial ratio r , then there is a solution s' in the population such that $|\mu(s')| = |\mu(s)|$, and the partial ratio of s' is at most r .*

Proof. s is put into the population after it is generated; otherwise there is another solution s' with $|\mu(s')| = |\mu(s)|$ and $f(s') \leq f(s)$, and in this case let $s = s'$. The lemma is proved because $\mathcal{L}(\mu(s)) = \mathcal{L}(\mu(s'))$ and by the superior function, the function f is non-increasing. \square

Thus, Lemma 6.1 shows that the partial ratio in each isolation is non-increasing. As SEIP tries to repeatedly generate solutions in each isolation, SEIP can be considered as optimizing the partial ratio in each isolation.

Lemma 6.2. *Given a minimization problem (n, f, \mathcal{C}) , an isolation function μ and its partial reference function \mathcal{L} , for three (partial) solutions s, s' and z such that $z = s \cup s'$, we have*

$$\frac{f(z)}{\mathcal{L}(\mu(z))} \leq \max \left\{ \frac{f(s)}{\mathcal{L}(\mu(s))}, \frac{f(s' | s)}{\mathcal{L}(\mu(s') | \mu(s))} \right\}. \quad (6.4)$$

Proof. Considering $z = s \cup s'$, by definition, we have

$$f(z) = f(s) + f(s' | s), \quad (6.5)$$

$$\mu(z) = \mu(s \cup s') = \mu(s) \cup \mu(s'). \quad (6.6)$$

Thus, we have

$$\begin{aligned} \frac{f(z)}{\mathcal{L}(\mu(z))} &= \frac{f(s) + f(s' | s)}{\mathcal{L}(\mu(s) \cup \mu(s'))} \\ &= \frac{f(s) + f(s' | s)}{\mathcal{L}(\mu(s)) + \mathcal{L}(\mu(s') | \mu(s))} \\ &\leq \max \left\{ \frac{f(s)}{\mathcal{L}(\mu(s))}, \frac{f(s' | s)}{\mathcal{L}(\mu(s') | \mu(s))} \right\}. \end{aligned} \quad (6.7)$$

□

Lemma 6.2 reveals that the partial ratio of a solution is related to the conditional partial ratio of a building block. This can be considered as the way how SEIP optimizes the partial ratio in each isolation, i.e., by optimizing the conditional partial ratio of each building-block partial solution. Theorem 6.1 shows the approximation ratio achieved by SEIP.

Theorem 6.1. *Given a minimization problem (n, f, \mathcal{C}) , an isolation function μ mapping to subsets of $[q]$ and its partial reference function \mathcal{L} , it is assumed that every solution is encoded in a length- n binary vector. If for all partial solutions s satisfying*

$$\frac{f(s)}{\mathcal{L}(\mu(s))} \leq r \quad (6.8)$$

(including the empty solution s^\emptyset), SEIP takes s as the parent solution and generates an offspring solution $s \cup s'$ holding that $\mu(s) \subset \mu(s \cup s')$ and

$$\frac{f(s' | s)}{\mathcal{L}(\mu(s') | \mu(s))} \leq r \quad (6.9)$$

in polynomial time in q and n , then SEIP finds an r -approximate solution in polynomial time in q and n .

Proof. Starting from the empty solution s^\emptyset , we can find a sequence of partial solutions s^1, s^2, \dots, s^m , such that

$$s = \cup_{i=1}^m s^i \text{ is feasible,} \quad (6.10)$$

and

$$\forall i \in [m] : \frac{f(s^i \mid s^\emptyset \cup (\cup_{j=1}^{i-1} s^j))}{\mathcal{L}(\mu(s^i) \mid \mu(s^\emptyset \cup (\cup_{j=1}^{i-1} s^j)))} \leq r, \quad (6.11)$$

due to the condition. Note that when a partial solution is added into the solution, the offspring solution is in a different isolation to the parent solution. Because the isolation function is linearly additive, the length m of the sequence cannot be greater than q .

Let t denote the expected running time, which is polynomial in q and n , for SEIP to generate a partial solution s^i in the sequence from its parent solution. It takes $O(q)$ expected steps for SEIP to pick the parent solution, because there are at most $(q + 1)$ solutions in the population. Therefore, the total time to reach the feasible solution s is $O(t \cdot q \cdot m) = O(tq^2)$, still polynomial in q and n .

By Lemma 6.2, as the feasible solution s is composed by s^\emptyset and partial solutions s^1, s^2, \dots, s^m , the approximation ratio of s is at most as large as the maximum conditional partial ratio of the partial solutions, i.e., r . \square

Theorem 6.1 discloses how SEIP works to achieve an approximate solution. Starting from the empty set, SEIP uses the mutation operator to generate solutions in all isolations, and finally generates a feasible solution. During the process, SEIP repeats to optimize the partial ratio in each isolation, by finding partial solutions with better conditional partial ratios. The feasible solution can be viewed as a composition of a sequence of partial solutions from the empty set, and thus, the approximation ratio is related to the conditional partial ratio of each partial solution.

In Theorem 6.1, the approximation ratio is upper bounded by the maximum conditional partial ratio, whereas some partial solutions with low conditional partial ratios are not utilized. Also, in Theorem 6.1 we restrict SEIP to append partial solutions, whereas GSEIP can remove partial solutions by bit-wise mutation. The approximation ratio can have a tighter bound if we consider relaxations on these two issues. With the same principle of Theorem 6.1, we present a theorem particularly for GSEIP which leads to a tighter approximation ratio.

Definition 6.4 (Non-negligible Path). *Given a minimization problem (n, f, \mathcal{C}) , an isolation function μ mapping to subsets of $[q]$ and its partial reference function \mathcal{L} , it is assumed that every solution is encoded in a length- n binary vector. A set P of solutions is a non-negligible path with ratios $\{r_i\}_{i=0}^q$ and gap c , if it holds that*

1. $s^\emptyset \in P$;
2. if $s \in P$ and $\text{superior}(s', s) = \text{true}$, then $s' \in P$;
3. $\forall s \in P$ there exists a solution $(s \cup z^+ \setminus z^-) \in P$, where the pair of solutions (z^+, z^-) satisfies that

$$(1) \quad 1 \leq |z^+|_1 + |z^-|_1 \leq c, \quad (6.12)$$

$$(2) \quad f(s \cup z^+ \setminus z^-) - f(s) \leq r_{|\mu(s)|} \cdot \text{OPT}, \quad (6.13)$$

$$(3) \quad \text{if } |\mu(s)| < q, \text{ then } |\mu(s \cup z^+ \setminus z^-)| > |\mu(s)|. \quad (6.14)$$

Theorem 6.2. Given a minimization problem (n, f, \mathcal{C}) , an isolation function μ mapping to subsets of $[q]$ and its partial reference function \mathcal{L} , it is assumed that every solution is encoded in a length- n binary vector. If there exists a non-negligible path with ratios $\{r_i\}_{i=0}^q$ and gap c , then GSEIP finds a $(\sum_{i=0}^{q-1} r_i)$ -approximate solution in $O(q^2 n^c)$ expected time.

Proof. We prove the theorem by tracking the process of GSEIP following the non-negligible path. Denote s^{cur} as the solution to be operated. Initially, $s^{\text{cur}} = s^\emptyset$, and thus, $|\mu(s^{\text{cur}})| = 0$.

GSEIP takes $O(q)$ expected steps to operate on s^{cur} , because there are at most $(q + 1)$ solutions in the population and GSEIP selects one to operate in each step. By the definition of non-negligible path, there exists a pair of solutions (z^+, z^-) with respect to s^{cur} , such that $1 \leq |z^+|_1 + |z^-|_1 \leq c$. Denote $s' = s^{\text{cur}} \cup z^+ \setminus z^-$. The probability for the bit-wise mutation operator to generate the solution s' is at least $(1/n)^c (1 - 1/n)^{n-c}$, implying $O(n^c)$ expected steps.

By the definition of non-negligible path, suppose $|\mu(s^{\text{cur}})| = i$, we have $f(s^{\text{cur}} \cup z^+ \setminus z^-) - f(s^{\text{cur}}) \leq r_i \cdot \text{OPT}$. Because $f(s')$ can be decomposed recursively by the condition of the theorem, we have

$$\begin{aligned} f(s') &= f(s^{\text{cur}} \cup z^+ \setminus z^-) - f(s^{\text{cur}}) + f(s^{\text{cur}}) \\ &\leq r_i \cdot \text{OPT} + f(s^{\text{cur}}) \leq \dots \\ &\leq \sum_{j=0}^i r_j \cdot \text{OPT} + f(s^\emptyset) = \sum_{j=0}^i r_j \cdot \text{OPT}. \end{aligned} \quad (6.15)$$

Thus, the partial ratio of s' is at most $\sum_{j=0}^i r_j \cdot \text{OPT}/\mathcal{L}(\mu(s'))$.

Given $|\mu(s^{\text{cur}})| = i$, again by the definition of non-negligible path, we have $|\mu(s')| > |\mu(s^{\text{cur}})|$. Then, we store the solution s' into the population. Otherwise there will exist another solution s'' satisfying that $|\mu(s'')| = |\mu(s')|$ and s'' has a no larger partial ratio than s' by Lemma 6.1; in this case, we substitute s'' for s' . Now let s^{cur} be s' . The partial ratio of s^{cur} is at most $\sum_{j=0}^{|\mu(s^{\text{cur}})|-1} r_j \cdot \text{OPT}/\mathcal{L}(\mu(s^{\text{cur}}))$.

After at most q iterations of the above update of s^{cur} , we have $|\mu(s^{\text{cur}})| = q$, implying that s^{cur} is feasible. The partial ratio of s^{cur} is now its approximation ratio

$$\frac{\sum_{j=0}^{q-1} r_j \cdot \text{OPT}}{\mathcal{L}(\mu(s^{\text{cur}}))} = \frac{\sum_{j=0}^{q-1} r_j \cdot \text{OPT}}{\text{OPT}} = \sum_{j=0}^{q-1} r_j. \quad (6.16)$$

Thus, at most q jumps are required to reach a feasible solution. Each jump takes $O(n^c)$ expected steps for operation on a particular solution, and takes $O(q)$ expected steps to select the particular solution. Overall, it takes $O(q^2 n^c)$ expected steps to achieve the feasible solution with an approximation ratio of $\sum_{j=0}^{q-1} r_j$. \square

Using Theorem 6.2 to prove the approximation ratio of GSEIP on a specific problem, we need to find one non-negligible path and then calculate the conditional evaluation function for each jump on the path. One way to find a non-negligible path for a problem is to follow an existing algorithm for the problem. This will lead to a proof that the EA can achieve the same approximation ratio by simulating the existing algorithm. Similar ideas have been used to confirm that EAs can simulate dynamic programming [Doerr et al., 2011a]. Note that the concrete form of the partial reference function is not required in the theorem.

6.2 SEIP: Application Illustration

In this section, we apply SEIP to the set cover problem as an illustration.

6.2.1 Set Cover

Definition 6.5 (Set Cover). Given a ground set $U = \{e_1, e_2, \dots, e_m\}$ of m elements, and a collection $V = \{S_1, S_2, \dots, S_n\}$ of n non-empty subsets of U , where each S_i is associated with a positive weight $w(S_i)$, to find a subset $X^* \subseteq V$ such that $\sum_{S \in X^*} w(S)$ is minimized with respect to $\cup_{S \in X^*} S = U$.

This is an NP-hard problem [Feige, 1998]. By binary vector representation, an instance of the set cover problem is denoted by its parameters (n, \mathbf{w}, V, U) , where the cardinality of V is n and \mathbf{w} is the weight vector. The set cover problem is to find a vector s^{OPT} , equivalent to its set representation X^* , such that

$$\begin{aligned} s^{\text{OPT}} &= \arg \min_{s \in \{0,1\}^n} \mathbf{w} \cdot s \\ \text{s.t. } &\cup_{i:s_i=1} S_i = U, \end{aligned} \quad (6.17)$$

where $\mathbf{w} \cdot s$ is the inner product between vectors \mathbf{w} and s .

The greedy algorithm [Chvátal, 1979] in Algorithm 6.2 is the most famous approximation algorithm for the set cover problem. It consists of a

Algorithm 6.2 Greedy Algorithm**Input:** set cover problem (n, \mathbf{w}, V, U) **Output:** set $X \subseteq V$ with $\cup_{S \in X} S = U$ **Process:**

- 1: let $X = \emptyset$ and $R = \emptyset$;
- 2: **while** $R \neq U$ **do**
- 3: $\forall S \in V : |S \setminus R| > 0$, let $r_S = w(S)/|S \setminus R|$;
- 4: $\hat{S} = \arg \min_S r_S$;
- 5: $\forall e \in \hat{S} \setminus R$, let $price(e) = r_{\hat{S}}$;
- 6: $X = X \cup \{\hat{S}\}$ and $R = R \cup \hat{S}$
- 7: **end while**
- 8: **return** X

sequence of steps. In each step, the cost of a candidate set, i.e., the quantity r_S in line 3 of Algorithm 6.2, is defined as its weight divided by the number of its elements that have not been covered yet. The algorithm then picks the candidate set with the smallest cost for the solution in line 4 and marks the newly covered elements in line 6. This simple algorithm yields an H_m -approximation ratio, or more exactly H_k where k is the cardinality of the largest set in V [Chvátal, 1979]. The key to the proof of the approximation ratio is the definition of the *price* of elements in line 5. The *price* of an element is equal to the cost of the set which first covers it, and therefore, the total *price* of all elements is equal to the total weight of the solution. Furthermore, when an element is covered by a set with the lowest cost, it would also be covered by one set of an optimal solution but with a higher cost. Therefore, the *price* of the element is upper bounded by the optimal cost, and hence, the approximation ratio is upper bounded. For a detailed proof please refer to [Chvátal, 1979].

Several studies [Raz and Safra, 1997, Slavík, 1997, Feige, 1998, Alon et al., 2006] have shown that the approximation ratio of the set cover problem is lower bounded by $\Omega(\log m)$ unless $P = NP$. Therefore, the greedy algorithm achieves the asymptotic lower bound of the approximation ratio on the set cover problem.

6.2.2 Approximation Ratios of SEIP

To apply SEIP to the set cover problem, we use the fitness function

$$f(\mathbf{s}) = \mathbf{w} \cdot \mathbf{s}, \quad (6.18)$$

which is the objective of minimizing the total weight. For a solution \mathbf{s} , denote $R(\mathbf{s}) = \cup_{i:s_i=1} S_i$, i.e., $R(\mathbf{s})$ is the set of elements covered by \mathbf{s} . We use the isolation function

$$\mu(\mathbf{s}) = R(\mathbf{s}), \quad (6.19)$$

which will make two solutions compete only when they cover the same number of elements. We can regard the partial reference function \mathcal{L} of s as the minimum price for optimal solutions to pay for covering the same number of elements covered by s , though it does not require to calculate the partial reference function for any solution in the analysis.

Theorem 6.3 shows that GSEIP achieves an H_k -approximation ratio, where k is the cardinality of the largest set in V . It has been proved that a simple MOEA, i.e., SEMO in Algorithm 2.5, achieves an H_m -approximation ratio for the set cover problem [Friedrich et al., 2010], known as the asymptotic lower bound for the problem. The theorem confirms that SEIP can simulate MOEAs in term of the solutions retained, so that SEIP is able to achieve the approximation ratio obtained by MOEAs.

The theorem is proved by applying Theorem 6.2. The intuition of the proof is to simulate the greedy algorithm. The proof uses the property of the greedy algorithm in Lemma 6.3, which is derived from [Chvátal, 1979]. For an optimal solution s^{OPT} , or equivalently X^* , let $X^*(e)$ denote a set in X^* which contains an element e .

Lemma 6.3. *Given a set cover problem (n, w, V, U) and an arbitrary partial solution s , let $\hat{S} = \arg \min_{S \in V} r_S$ with respect to s . For every element $e \in \hat{S} \setminus R(s)$, there exists a set $X^*(e)$ in an optimal solution which covers e , and*

$$\text{price}(e) \leq \frac{w(X^*(e))}{|X^*(e) \setminus R(s)|}. \quad (6.20)$$

Theorem 6.3. *Given a set cover problem (n, w, V, U) where $|V| = n$ and $|U| = m$, GSEIP finds an H_k -approximate solution in $O(nm^2)$ expected running time, where k is the size of the largest set in V .*

Proof. We find a non-negligible path by following the greedy rule, i.e., given the current partial solution, add the set \hat{S} with the minimum r_S as in lines 3-6 of Algorithm 6.2.

Denote s^{cur} as the current solution to be operated. We find $\mathbf{z}^+ = \{\hat{S}\}$ where \hat{S} minimizes r_S with respect to s^{cur} . Let $\mathbf{z}^- = \emptyset$. Thus, we have $|\mathbf{z}^+|_1 + |\mathbf{z}^-|_1 = 1$ and $|\mu(s^{\text{cur}} \cup \mathbf{z}^+ \setminus \mathbf{z}^-)| \geq |\mu(s^{\text{cur}})| + 1$ for the partial solution s^{cur} .

By Lemma 6.3, $\forall e \in \hat{S} \setminus R(s^{\text{cur}})$, there exists a set $X^*(e)$ in an optimal solution which covers e and

$$\text{price}(e) \leq \frac{w(X^*(e))}{|X^*(e) \setminus R(s^{\text{cur}})|}. \quad (6.21)$$

According to the definition of $\text{price}(e)$, we have

$$\begin{aligned} f(s^{\text{cur}} \cup \mathbf{z}^+ \setminus \mathbf{z}^-) - f(s^{\text{cur}}) &= w(\hat{S}) = \sum_{e \in \hat{S} \setminus R(s^{\text{cur}})} \text{price}(e) \\ &\leq \sum_{e \in \hat{S} \setminus R(s^{\text{cur}})} \frac{w(X^*(e))}{|X^*(e) \setminus R(s^{\text{cur}})|}. \end{aligned} \quad (6.22)$$

Denote an optimal solution X^* as $\{S_1^*, \dots, S_{|X^*|}^*\}$. Let $s^{\text{cur}}(e)$ denote the partial solution which will cover e in its next step. Thus, we find a non-negligible path with gap 1 and sum of ratios,

$$\begin{aligned} \sum_{e \in U} \frac{w(X^*(e))}{|X^*(e) \setminus R(s^{\text{cur}}(e))|} &\leq \sum_{j=1}^{|X^*|} \sum_{e \in S_j^*} \frac{w(S_j^*)}{|S_j^* \setminus R(s^{\text{cur}}(e))|} \leq \sum_{j=1}^{|X^*|} w(S_j^*) \cdot \sum_{i=1}^k \frac{1}{i} \\ &= \sum_{j=1}^{|X^*|} w(S_j^*) \cdot H_k = H_k \cdot \text{OPT}. \end{aligned} \quad (6.23)$$

By Theorem 6.2, GSEIP finds an H_k -approximate solution. Note that the isolation function maps to at most $(m + 1)$ isolations, the non-negligible path is with gap 1, and the solution is encoded in a length- n binary vector; thus GSEIP takes $O(nm^2)$ expected running time. \square

In the proof of Theorem 6.3, with respect to s^{cur} , we find z^+ and z^- , satisfying that $|z^+|_1 = 1$ and $|z^-|_1 = 0$. Thus, the proof can be adapted to LSEIP directly, leading to Theorem 6.4.

Theorem 6.4. *Given a set cover problem (n, w, V, U) where $|V| = n$ and $|U| = m$, LSEIP finds an H_k -approximate solution in $O(nm^2)$ expected running time, where k is the size of the largest set in V .*

6.3 Summary

In this chapter, we study the approximation performance of EAs by analyzing the general SEIP framework. SEIP adopts an isolation function to manage competition among solutions, which can be configured as single- or multi-objective EAs. We obtain a general characterization of approximation behaviors of SEIP using the concept of partial ratio. Our analysis discloses how the behaviors of SEIP lead to guaranteed approximation solutions: they try to optimize the partial ratio of solutions in every isolation by finding good partial solutions, and then these partial solutions compose a feasible solution with a good approximation ratio. As an application illustration, we show that for the set cover problem, SEIP can achieve an H_m -approximation ratio, the asymptotic lower bound.

To prove the approximation ratio of SEIP on the set cover problem, we use SEIP to simulate the greedy algorithm. As the greedy algorithm is a general scheme for approximation and has been analyzed for many problems, there is a potential to extend the analysis of SEIP to cases for which the greedy algorithm has been applied. For example, for k -extensible systems, a general problem class including *b-matching*, *maximum profit scheduling* and the maximum asymmetric *travelling salesman problem* (TSP), the greedy algorithm is an $(1/k)$ -approximation algorithm [Mestre, 2006] and we can show that SEIP also achieves an approximation ratio of $1/k$.

Part III

THEORETICAL PERSPECTIVES



Boundary Problems of EAs

This chapter studies a natural theoretical question; that is, how well EAs can perform on classes of problems. Previous studies generally focused on restricted problem classes where the problem instances have similar structures. It becomes much more difficult when considering large problem classes, as the existence of a large variety of structures of problem instances makes the analysis more difficult.

One possible way to simplify the analysis over a problem class is to characterize the class by its easiest and hardest instances, and then analyze these boundary instances. By the well-known *no free lunch* theorem [Wolpert and Macready, 1997], the hardness has to be studied by considering the problem instance and the algorithm together. Thus, this chapter studies the algorithm-dependent boundary case identification problem, as illustrated in Figure 7.1. Given an EA, the identification of boundary instances of a problem class can not only help study the general performance of the algorithm over the problem class, but also provide concrete instances to reveal the strength and weakness of the concerned EA.

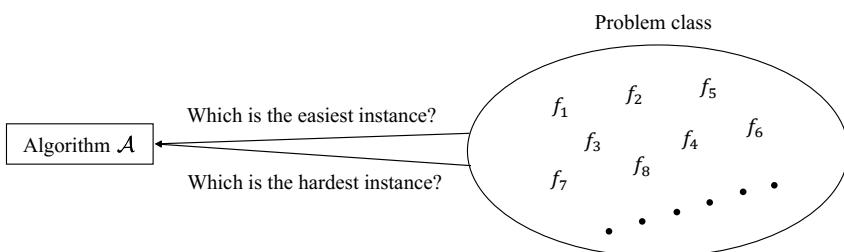


Figure 7.1. Algorithm-dependent boundary case identification.

For this purpose, we present a general theorem [Qian et al., 2012] for algorithm-dependent boundary case identification, which gives a sufficient

condition for identifying the easiest and hardest instances of a problem class for an algorithm. As an application illustration, we prove that in the UBoolean function class in Definition 2.2, i.e., the pseudo-Boolean function class with a unique global optimal solution, the OneMax and Trap problems are the easiest and hardest instances, respectively, for (1+1)-EA with any mutation probability less than 0.5.

There are a few studies concerning problem classes. The running time of EAs on the linear pseudo-Boolean function class, which is a relatively small problem class, was analyzed in [Droste et al., 2002, Jägersküpper, 2008, Doerr et al., 2012b]. Yu and Zhou [2008a] provided a general understanding on why EAs may fail over a complex problem class. Fournier and Teytaud [2011] provided a general lower bound for the performance of EAs over problem classes with complexity measured by VC-dimension. Friedrich and Neumann [2015] proved the approximation ratios of (1+1)-EA and GSEMO on the problem class of maximizing submodular functions, which contains many NP-hard combinatorial problems. These studies, however, did not concern the boundary problem instances. Recently, Doerr et al. [2010b] proved that OneMax is the easiest instance in the UBoolean function class for (1+1)-EA with the mutation probability $1/n$. In this chapter, we will derive general results for the easiest and hardest instances.

The rest of this chapter is organized as follows. Section 7.1 presents the main theorem for boundary case identification of a problem class. Section 7.2 applies the theorem to identify the boundary instances in the UBoolean class for (1+1)-EA. Section 7.3 concludes this chapter.

7.1 Boundary Problem Identification

We assume that the concerned problem class is homogeneous, as stated in Definition 7.1, which means that all problem instances in the class have the same solution space and the same target solutions when the problem dimensionality is fixed. At first glance, the requirement of the same target solutions is a strong restriction. However, this is an illusion, because most EAs do not rely on the semantics of the solution, and thus, the target solutions can be made to be the same by switching some bits; for example, the solution 10011 in a binary space can be shifted to 11111 if we switch the semantics of 1 and 0 for the second and third bits.

Definition 7.1 (Homogeneous Problem Class). *A problem class is homogeneous if, for each problem dimensionality, all problem instances have the same solution space and the same target solutions.*

Assume that EA processes can be modeled by homogeneous Markov chains. To characterize the algorithm-dependent structure of the problem instances, the state space of a homogeneous Markov chain is partitioned

based on the CFHT, as in Definition 7.2. It can be seen that \mathcal{X}_0 is just \mathcal{X}^* , because $\mathbb{E}[\tau \mid \xi_0 \in \mathcal{X}^*] = 0$.

Definition 7.2 (CFHT-Partition). For a homogeneous Markov chain $\{\xi_t\}_{t=0}^{+\infty}$ with state space \mathcal{X} , the CFHT-Partition is a partition of \mathcal{X} into non-empty subspaces $\{\mathcal{X}_0, \mathcal{X}_1, \dots, \mathcal{X}_m\}$ such that

$$(1) \quad \forall x, y \in \mathcal{X}_i : \mathbb{E}[\tau \mid \xi_0 = x] = \mathbb{E}[\tau \mid \xi_0 = y], \quad (7.1)$$

$$(2) \quad \mathbb{E}[\tau \mid \xi_0 \in \mathcal{X}_0] < \mathbb{E}[\tau \mid \xi_0 \in \mathcal{X}_1] < \dots < \mathbb{E}[\tau \mid \xi_0 \in \mathcal{X}_m]. \quad (7.2)$$

Note that the fitness level methods in Theorems 2.1 and 2.2 for EAs' running time analysis also use a way to partition a state space, i.e., the $<_f$ -partition in Definition 2.15. The CFHT-partition is based on the CFHT, whereas the $<_f$ -partition is based on the fitness of solutions. They are different because solutions with the same fitness can have different CFHTs, and the CFHT order can be either consistent, e.g., (1+1)-EA on the Trap problem, or inconsistent, e.g., (1+1)-EA on the OneMax problem, with the fitness order.

Lemma 7.1 compares the DCFHT of two homogeneous Markov chains. Intuitively, it means that if one chain always has a larger probability of jumping into good states, i.e., \mathcal{X}_j with small j values, it requires less time to reach the target state space.

Lemma 7.1. Given two homogeneous absorbing Markov chains $\{\xi_t\}_{t=0}^{+\infty}$ and $\{\xi'_t\}_{t=0}^{+\infty}$ with the same state space \mathcal{X} and the same target state subspace \mathcal{X}^* , let $\{\mathcal{X}_0, \mathcal{X}_1, \dots, \mathcal{X}_m\}$ denote the CFHT-Partition of $\{\xi'_t\}_{t=0}^{+\infty}$. If $\forall t \geq 0, x \in \mathcal{X} \setminus \mathcal{X}_0$, and $\forall i \in \{0, 1, \dots, m-1\}$,

$$\sum_{j=0}^i P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) \geq (\leq) \sum_{j=0}^i P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x), \quad (7.3)$$

then $\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \leq (\geq) \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0]$.

Proof. We use the switch analysis approach in Theorem 4.1 to prove one direction of Eq. (7.3), and the other can be proved similarly.

We denote the state space and target subspace of $\{\xi'_t\}_{t=0}^{+\infty}$ by \mathcal{Y} and \mathcal{Y}^* , respectively, where $\mathcal{Y} = \mathcal{X}$ and $\mathcal{Y}^* = \mathcal{X}^*$. We construct the mapping $\phi : \mathcal{X} \rightarrow \mathcal{Y}$ as $\forall x \in \mathcal{X} : \phi(x) = x$. It is clear that $\phi(x) \in \mathcal{Y}^*$ iff $x \in \mathcal{X}^*$.

Next we examine the condition, i.e., Eq. (4.1), of switch analysis. For a target state $x \in \mathcal{X}^* = \mathcal{X}_0$, because $\phi(x) = x$ and both Markov chains are absorbing, we have

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ &= \sum_{y \in \mathcal{Y}} P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] = 0. \end{aligned} \quad (7.4)$$

For a non-target state $x \in \mathcal{X}_i$ with $i \geq 1$, because $\phi(x) = x$ and $\{\xi'_t\}_{t=0}^{+\infty}$ is homogeneous, we have

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] \\ &= \sum_{j=0}^m P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x) \mathbb{E}[\tau' \mid \xi'_0 \in \mathcal{X}_j], \end{aligned} \quad (7.5)$$

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ &= \sum_{j=0}^m P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 \in \mathcal{X}_j]. \end{aligned} \quad (7.6)$$

To compare Eqs. (7.5) and (7.6), we apply Lemma 4.4. The conditions of Lemma 4.4 are verified because $\sum_{j=0}^m P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) = \sum_{j=0}^m P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x) = 1$; Eq. (7.3) holds, i.e., $\forall i \in \{0, 1, \dots, m-1\} : \sum_{j=0}^i P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) \geq \sum_{j=0}^i P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x)$; $\mathbb{E}[\tau' \mid \xi'_0 \in \mathcal{X}_j]$ increases with j . By Lemma 4.4, we have

$$\begin{aligned} & \sum_{j=0}^m P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x) \mathbb{E}[\tau' \mid \xi'_0 \in \mathcal{X}_j] \\ & \geq \sum_{j=0}^m P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 \in \mathcal{X}_j], \end{aligned} \quad (7.7)$$

and thus,

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ & \leq \sum_{y \in \mathcal{Y}} P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y]. \end{aligned} \quad (7.8)$$

By combining Eqs. (7.4) and (7.8), we have

$$\begin{aligned} & \forall t \geq 0 : \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_t(x) P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ & \leq \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_t(x) P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] \\ &= \sum_{u, y \in \mathcal{Y}} \pi_t^\phi(u) P(\xi'_1 = y \mid \xi'_0 = u) \mathbb{E}[\tau' \mid \xi'_1 = y], \end{aligned} \quad (7.9)$$

where Eq. (7.9) holds by $\pi_t^\phi(u) = \pi_t(\phi^{-1}(u))$.

Thus, Eq. (4.1) holds with $\rho_t = 0$. By switch analysis, we have

$$\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \leq \mathbb{E}[\tau' \mid \xi'_0 \sim \pi'_0]. \quad (7.10)$$

Because $\pi_0 = \pi'_0$, we have $\mathbb{E}[\tau_0 \mid \xi_0 \sim \pi_0] \leq \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0]$. \square

We then present Theorem 7.1, which gives a general sufficient condition for identifying the easiest and hardest problem instances. Note that, the easiest (hardest) problem instance of a problem class for an algorithm implies that the expected running time of the algorithm on the problem instance is the least (greatest).

Theorem 7.1. *Given a homogeneous problem class \mathcal{F} and an algorithm \mathcal{A} , with dimensionality n , let $\{\xi'_t\}_{t=0}^{+\infty}$ model \mathcal{A} running on a problem $f^* \in \mathcal{F}_n$, of which $\{\mathcal{X}_0, \mathcal{X}_1, \dots, \mathcal{X}_m\}$ is the CFHT-Partition. If $\forall f \in \mathcal{F}_n \setminus \{f^*\}$, $\forall t \geq 0$, and $\forall x \in \mathcal{X} \setminus \mathcal{X}_0$, denoting $\{\xi_t\}_{t=0}^{+\infty}$ as the chain modeling \mathcal{A} running on f , $\exists k \in \{0, 1, \dots, m\}$,*

$$\forall j \leq k : P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) \leq (\geq) P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x), \quad (7.11)$$

$$\forall j > k : P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) \geq (\leq) P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x), \quad (7.12)$$

then f^* is the easiest (hardest) instance in \mathcal{F}_n for the algorithm \mathcal{A} .

Proof. We use Lemma 7.1 to show the easiest problem instance identification of this theorem by proving that the expected running time of the algorithm \mathcal{A} on the problem f^* is no greater than that on any other problem. The hardest instance identification can be proved similarly. Note that as introduced in Section 2.4, both Markov chains $\{\xi_t\}_{t=0}^{+\infty}$ and $\{\xi'_t\}_{t=0}^{+\infty}$ can be transformed to be absorbing without affecting their first hitting time.

The two chains $\{\xi_t\}_{t=0}^{+\infty}$ and $\{\xi'_t\}_{t=0}^{+\infty}$ have the same state space \mathcal{X} and the same target subspace \mathcal{X}^* , because the concerned problem class is homogeneous. By Eqs. (7.11) and (7.12), we can derive that $\forall i \in \{0, 1, \dots, m-1\}$,

$$\sum_{j=0}^i P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) \leq \sum_{j=0}^i P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x), \quad (7.13)$$

implying that the condition, i.e., Eq. (7.3), of Lemma 7.1 holds. Thus, we have $\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \geq \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0]$. The two sides of this inequality are the DCFHT of two chains, one modeling \mathcal{A} on a problem $f \in \mathcal{F}_n \setminus \{f^*\}$ and the other on the problem f^* . This inequality holds $\forall f \in \mathcal{F}_n \setminus \{f^*\}$. Thus, f^* is the easiest problem instance in \mathcal{F}_n for the algorithm \mathcal{A} . \square

7.2 Case Study

In this section, we use Theorem 7.1 to identify the easiest and hardest functions in the UBoolean function class for (1+1)-EA with any mutation probability less than 0.5.

The pseudo-Boolean function class in Definition 2.1 is a large function class which only requires the solution space to be $\{0, 1\}^n$ and the objective space to be \mathbb{R} . The UBoolean function class in Definition 2.2 is the pseudo-Boolean function class with a unique global optimal solution, which is a subset of the pseudo-Boolean function class. In other words, the global optimal solution of each function in UBoolean is unique. Here, maximization problems are considered, as minimizing f is equivalent to maximizing $-f$. For UBoolean, as introduced in Section 2.2, we can assume without loss of generality that the optimal solution of any function in UBoolean is 1^n .

We prove in Theorem 7.2 that the OneMax and Trap problems are the easiest and hardest instances, respectively, in UBoolean for (1+1)-EA. The OneMax problem in Definition 2.3 is to maximize the number of 1-bits of a solution. The Trap problem in Definition 2.6 is to maximize the number of 0-bits of a solution except for the global optimal solution 1^n . They are two special instances in the UBoolean function class.

Before giving the proof, we need to prove the order of the CFHT of (1+1)-EA on the OneMax problem as well as that on the Trap problem. For (1+1)-EA on the OneMax problem, Lemma 4.2 shows that the CFHT $\mathbb{E}[\tau' | \xi'_t = x]$ with $|x|_0 = j$ can be denoted by $\mathbb{E}'(j)$ which increases with j . For the Trap problem, the CFHT $\mathbb{E}[\tau' | \xi'_t = x]$ of (1+1)-EA depends only on the number of 0-bits of the solution x . Thus, we denote $\mathbb{E}'(j)$ as the CFHT $\mathbb{E}[\tau' | \xi'_0 = x]$ with $|x|_0 = j$. $\mathbb{E}'(0) = 0$ implies the optimal solution. The order of $\mathbb{E}'(j)$, which increases with j , is shown in Lemma 7.2.

Lemma 7.2. *For any mutation probability $p \in (0, 0.5)$, it holds that*

$$\mathbb{E}'(0) < \mathbb{E}'(1) < \mathbb{E}'(2) < \cdots < \mathbb{E}'(n). \quad (7.14)$$

Proof. It holds that $\mathbb{E}'(0) < \mathbb{E}'(1)$, because $\mathbb{E}'(0) = 0$ and $\mathbb{E}'(1) > 0$. Next, we prove $\forall 0 < j < n : \mathbb{E}'(j) < \mathbb{E}'(j + 1)$ inductively on j .

(a) Initialization is to prove $\mathbb{E}'(n - 1) < \mathbb{E}'(n)$. For $\mathbb{E}'(n)$, because only the offspring solution 0^n or 1^n will be accepted, we have $\mathbb{E}'(n) = 1 + p^n \mathbb{E}'(0) + (1 - p^n) \mathbb{E}'(n)$, leading to $\mathbb{E}'(n) = 1/p^n$. For $\mathbb{E}'(n - 1)$, because the accepted offspring solutions are $0^n, 1^n$ and the solutions with $(n-1)$ number of 0-bits, we have $\mathbb{E}'(n - 1) = 1 + p^{n-1}(1-p)\mathbb{E}'(0) + p(1-p)^{n-1}\mathbb{E}'(n) + (1-p^{n-1}(1-p) - p(1-p)^{n-1})\mathbb{E}'(n - 1)$, leading to $\mathbb{E}'(n - 1) = (1 + (1-p)^{n-1}/p^{n-1})/(p^{n-1}(1-p) + p(1-p)^{n-1})$. Thus, we have

$$\frac{\mathbb{E}'(n)}{\mathbb{E}'(n - 1)} = \frac{p^{n-1}(1-p) + p(1-p)^{n-1}}{p^n + (1-p)^{n-1}p} > 1, \quad (7.15)$$

where the inequality holds by $0 < p < 0.5$.

(b) Inductive Hypothesis assumes that

$$\forall K < j \leq n - 1 (K \geq 1) : \mathbb{E}'(j) < \mathbb{E}'(j + 1). \quad (7.16)$$

Consider $j = K$. When comparing $\mathbb{E}'(K + 1)$ with $\mathbb{E}'(K)$, we use the same analysis method as that in the proof of Lemma 4.2. Let x and x' be a

solution with $(K + 1)$ number of 0-bits and that with K number of 0-bits, respectively. We have $\mathbb{E}'(K + 1) = \mathbb{E}[\tau' \mid \xi'_t = x]$ and $\mathbb{E}'(K) = \mathbb{E}[\tau' \mid \xi'_t = x']$. For a Boolean string of length $(n - 1)$ with K number of 0-bits, we denote P_i ($0 \leq i \leq n - 1$) as the probability for the number of 0-bits to become i after bit-wise mutation on this string with the mutation probability p .

For the solution x , the influence of mutation is divided into two parts: mutation on one 0-bit and mutation on the remaining $(n - 1)$ bits. The remaining $(n - 1)$ bits contain K number of 0-bits. By considering the mutation and selection behavior of (1+1)-EA on the Trap problem, we have

$$\begin{aligned} \mathbb{E}'(K + 1) &= 1 + p \cdot \left(P_0 \mathbb{E}'(0) + \sum_{i=1}^K P_i \mathbb{E}'(K + 1) + \sum_{i=K+1}^{n-1} P_i \mathbb{E}'(i) \right) \\ &\quad + (1 - p) \cdot \left(\sum_{i=0}^{K-1} P_i \mathbb{E}'(K + 1) + \sum_{i=K}^{n-1} P_i \mathbb{E}'(i + 1) \right), \end{aligned} \quad (7.17)$$

where the term p is the probability for the 0-bit in the first mutation part to be flipped.

For the solution x' , the influence of mutation is also divided into two parts: mutation on one 1-bit and mutation on the remaining $(n - 1)$ bits. The remaining $(n - 1)$ bits contain K number of 0-bits. Thus, we have

$$\begin{aligned} \mathbb{E}'(K) &= 1 + p \cdot \left(\sum_{i=0}^{K-2} P_i \mathbb{E}'(K) + \sum_{i=K-1}^{n-1} P_i \mathbb{E}'(i + 1) \right) \\ &\quad + (1 - p) \cdot \left(P_0 \mathbb{E}'(0) + \sum_{i=1}^{K-1} P_i \mathbb{E}'(K) + \sum_{i=K}^{n-1} P_i \mathbb{E}'(i) \right), \end{aligned} \quad (7.18)$$

where the term p is the probability for the 1-bit in the first mutation part to be flipped.

By comparing Eqs. (7.17) and (7.18), we have

$$\begin{aligned} &\mathbb{E}'(K + 1) - \mathbb{E}'(K) \\ &= p \left(P_0 (\mathbb{E}'(0) - \mathbb{E}'(K)) + \sum_{i=1}^{K-1} P_i (\mathbb{E}'(K + 1) - \mathbb{E}'(K)) \right. \\ &\quad \left. + \sum_{i=K+1}^{n-1} P_i (\mathbb{E}'(i) - \mathbb{E}'(i + 1)) \right) \\ &\quad + (1 - p) \left(P_0 (\mathbb{E}'(K + 1) - \mathbb{E}'(0)) + \sum_{i=1}^K P_i (\mathbb{E}'(K + 1) - \mathbb{E}'(K)) \right. \\ &\quad \left. + \sum_{i=K+1}^{n-1} P_i (\mathbb{E}'(i + 1) - \mathbb{E}'(i)) \right) \end{aligned}$$

$$\begin{aligned}
&= P_0((1-p)\mathbb{E}'(K+1) - p\mathbb{E}'(K)) \\
&\quad + \left(\sum_{i=1}^{K-1} P_i + (1-p)P_K \right) (\mathbb{E}'(K+1) - \mathbb{E}'(K)) \\
&\quad + (1-2p) \left(\sum_{i=K+1}^{n-1} P_i (\mathbb{E}'(i+1) - \mathbb{E}'(i)) \right) \\
&> \left(\sum_{i=1}^{K-1} P_i + (1-p)P_K + pP_0 \right) \cdot (\mathbb{E}'(K+1) - \mathbb{E}'(K)),
\end{aligned} \tag{7.19}$$

where Eq. (7.19) holds by $0 < p < 0.5$ and inductive hypothesis.

Because $\sum_{i=1}^{K-1} P_i + (1-p)P_K + pP_0 < 1$, we have $\mathbb{E}'(K+1) > \mathbb{E}'(K)$.
(c) Concluding from (a) and (b), the lemma holds. \square

Theorem 7.2. *In the UBoolean function class, the OneMax and Trap problems are the easiest and hardest problem instances, respectively, for (1+1)-EA with any mutation probability $p \in (0, 0.5)$.*

Proof. The UBoolean function class is homogeneous, because, for each dimensionality n , the solution space and the optimal solution for any function are $\{0, 1\}^n$ and 1^n , respectively. By the behavior of (1+1)-EA, clearly it can be modeled by a homogeneous Markov chain.

Let the OneMax problem correspond to f^* in Theorem 7.1. For the parameters m and \mathcal{X}_i in Theorem 7.1, we have $m = n$ and $\mathcal{X}_i = \{x \mid |x|_0 = i\}$ $\forall 0 \leq i \leq n$ according to Lemma 4.2. For any non-optimal solution $x \in \mathcal{X}_k$ with $k > 0$, let P_j ($0 \leq j \leq n$) denote the probability for the offspring solution generated by bit-wise mutation on x to have j number of 0-bits. For $\{\xi'_t\}_{t=0}^{+\infty}$, because only the offspring solution with no more 0-bits will be accepted, we have

$$\forall 0 \leq j \leq k-1 : P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x) = P_j, \tag{7.20}$$

$$P(\xi'_{t+1} \in \mathcal{X}_k \mid \xi'_t = x) = \sum_{j=k}^n P_j, \tag{7.21}$$

$$\forall k+1 \leq j \leq n : P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x) = 0. \tag{7.22}$$

For $\{\xi_t\}_{t=0}^{+\infty}$, because the offspring solution with less 0-bits may be rejected and that with more 0-bits may be accepted, we have

$$P(\xi_{t+1} \in \mathcal{X}_0 \mid \xi_t = x) = P_0, \tag{7.23}$$

$$\forall 1 \leq j \leq k-1 : P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) \leq P_j, \tag{7.24}$$

$$\forall k+1 \leq j \leq n : P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) \geq 0. \tag{7.25}$$

If $P(\xi'_{t+1} \in \mathcal{X}_k \mid \xi'_t = x) \geq P(\xi_{t+1} \in \mathcal{X}_k \mid \xi_t = x)$, we have

$$\forall 0 \leq j \leq k : P(\xi_{t+1} \in \mathcal{X}_j | \xi_t = x) \leq P(\xi'_{t+1} \in \mathcal{X}_j | \xi'_t = x), \quad (7.26)$$

$$\forall k+1 \leq j \leq n : P(\xi_{t+1} \in \mathcal{X}_j | \xi_t = x) \geq P(\xi'_{t+1} \in \mathcal{X}_j | \xi'_t = x). \quad (7.27)$$

Otherwise, we have

$$\forall 0 \leq j \leq k-1 : P(\xi_{t+1} \in \mathcal{X}_j | \xi_t = x) \leq P(\xi'_{t+1} \in \mathcal{X}_j | \xi'_t = x), \quad (7.28)$$

$$\forall k \leq j \leq n : P(\xi_{t+1} \in \mathcal{X}_j | \xi_t = x) \geq P(\xi'_{t+1} \in \mathcal{X}_j | \xi'_t = x). \quad (7.29)$$

By Theorem 7.1, the OneMax problem is the easiest in UBoolean for (1+1)-EA with any mutation probability $p \in (0, 0.5)$.

Let the Trap problem correspond to f^* . By Lemma 7.2, we have $m = n$ and $\mathcal{X}_i = \{x \mid |x|_0 = i\} \forall 0 \leq i \leq n$. For any non-optimal solution $x \in \mathcal{X}_k$ with $k > 0$, we also denote P_j ($0 \leq j \leq n$) as the probability for the offspring solution generated by bit-wise mutation on x to have j number of 0-bits. For $\{\xi'_t\}_{t=0}^{+\infty}$, because only the optimal solution and the offspring solutions with no less 0-bits will be accepted, we have

$$P(\xi'_{t+1} \in \mathcal{X}_0 | \xi'_t = x) = P_0, \quad (7.30)$$

$$\forall 1 \leq j \leq k-1 : P(\xi'_{t+1} \in \mathcal{X}_j | \xi'_t = x) = 0, \quad (7.31)$$

$$P(\xi'_{t+1} \in \mathcal{X}_k | \xi'_t = x) = \sum_{j=1}^k P_j, \quad (7.32)$$

$$\forall k+1 \leq j \leq n : P(\xi'_{t+1} \in \mathcal{X}_j | \xi'_t = x) = P_j. \quad (7.33)$$

For $\{\xi_t\}_{t=0}^{+\infty}$, because the offspring solution with less 0-bits may be accepted and that with more 0-bits may be rejected, we have

$$P(\xi_{t+1} \in \mathcal{X}_0 | \xi_t = x) = P_0, \quad (7.34)$$

$$\forall 1 \leq j \leq k-1 : P(\xi_{t+1} \in \mathcal{X}_j | \xi_t = x) \geq 0, \quad (7.35)$$

$$\forall k+1 \leq j \leq n : P(\xi_{t+1} \in \mathcal{X}_j | \xi_t = x) \leq P_j. \quad (7.36)$$

If $P(\xi'_{t+1} \in \mathcal{X}_k | \xi'_t = x) \geq P(\xi_{t+1} \in \mathcal{X}_k | \xi_t = x)$, we have

$$\forall 0 \leq j \leq k-1 : P(\xi_{t+1} \in \mathcal{X}_j | \xi_t = x) \geq P(\xi'_{t+1} \in \mathcal{X}_j | \xi'_t = x), \quad (7.37)$$

$$\forall k \leq j \leq n : P(\xi_{t+1} \in \mathcal{X}_j | \xi_t = x) \leq P(\xi'_{t+1} \in \mathcal{X}_j | \xi'_t = x). \quad (7.38)$$

Otherwise, we have

$$\forall 0 \leq j \leq k : P(\xi_{t+1} \in \mathcal{X}_j | \xi_t = x) \geq P(\xi'_{t+1} \in \mathcal{X}_j | \xi'_t = x), \quad (7.39)$$

$$\forall k+1 \leq j \leq n : P(\xi_{t+1} \in \mathcal{X}_j | \xi_t = x) \leq P(\xi'_{t+1} \in \mathcal{X}_j | \xi'_t = x). \quad (7.40)$$

By Theorem 7.1, the Trap problem is the hardest in UBoolean for (1+1)-EA with any mutation probability $p \in (0, 0.5)$. \square

7.3 Summary

In this chapter, we present a general theorem to identify the easiest and hardest problem instances of a problem class for an EA. Using the theorem, we prove that the OneMax and Trap problems are the easiest and hardest functions, respectively, in the UBoolean function class for (1+1)-EA with any mutation probability less than 0.5. This theorem can be applied to get results for more problem classes and more algorithms.



Recombination

Operator design has a direct influence on the performance of EAs. Note that existing operators, such as *mutation* and *recombination*, of EAs are often designed heuristically, lacking theoretical understanding. This chapter studies the effectiveness of recombination (or called *crossover*) operators in evolutionary multi-objective optimization.

A characterizing feature of EAs is the recombination operator for reproducing new solutions. Recombination operators take two or more individual solutions from the maintained population and mix them up to form new solutions. A commonly used one is one-point recombination shown in Figure 1.5, which exchanges the parts of two solutions separated by one randomly chosen position. Thus, recombination operators are population-based operators and typically appear in MOEAs, e.g., the popularly used algorithm NSGA-II [Deb et al., 2002]. The MOEAs that use recombination operators have been successful in solving various real-world problems, e.g., multiprocessor system-on-chip design [Erbas et al., 2006], aeronautical and aerospace engineering [Arias-Montano et al., 2012] and multicommodity capacitated network design [Kleeman et al., 2012]. However, recombination operators are understood only at a very preliminary level. Discovering the influence of recombination in MOEAs can not only enhance our understanding of this kind of nature-inspired operators, but also help design improved algorithms.

In this chapter, we present the performance comparison [Qian et al., 2013] between MOEAs using diagonal multi-parent recombination [Eiben et al., 1994], which is a generalization of one-point recombination over two solutions, and that using mutation only. We derive the expected running time of the MOEAs using recombination together with one-bit mutation and bit-wise mutation, denoted as $\text{MOEA}_{\text{recomb}}^{\text{onebit}}$ and $\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$, respectively. One-bit mutation implies a kind of local search, whereas bit-wise mutation can be viewed as a kind of global search. We indicate the probability of applying recombination by using the subscript following “recomb”, e.g., $\text{MOEA}_{\text{recomb},0.5}^{\text{onebit}}$ uses recombination with probability 0.5 and

$\text{MOEA}_{\text{recomb},0}^{\text{onebit}}$ does not use recombination. As EAs are a kind of optimization algorithms that are problem-independent, a typical way to study EAs theoretically is to use model problems to examine EAs' properties. In this chapter, we employ two bi-objective problems: the Weighted Leading Positive Ones Trailing Negative Ones (LPTNO) problem and the COCZ problem in Definition 2.8. Weighted LPTNO is generalized from the LOTZ problem in Definition 2.7.

On Weighted LPTNO and COCZ, we derive the expected running time of $\text{MOEA}_{\text{recomb},0.5}^{\text{onebit}}$ and $\text{MOEA}_{\text{recomb},0.5}^{\text{bitwise}}$ for finding the Pareto front. We compare them with that of $\text{MOEA}_{\text{recomb},0}^{\text{onebit}}$ and $\text{MOEA}_{\text{recomb},0}^{\text{bitwise}}$ as well as previously analyzed MOEAs without recombination on the LOTZ and COCZ problems, respectively. The results are summarized in Table 8.1, showing that $\text{MOEA}_{\text{recomb},0.5}^{\text{onebit}}$ and $\text{MOEA}_{\text{recomb},0.5}^{\text{bitwise}}$ achieve the best performance among MOEAs with one-bit mutation and bit-wise mutation, respectively. The theoretical results are further verified by experiments.

Table 8.1. Expected running time bounds of several MOEAs on LOTZ and COCZ, where the first column denotes the mutation operator used by the MOEAs in the corresponding row. Note that $\text{MOEA}^{\text{onebit}}$, $\text{MOEA}^{\text{onebit}}_{\text{fair}}$ and $\text{MOEA}^{\text{onebit}}_{\text{greedy}}$ are previously analyzed MOEAs without recombination operators, and their running time bounds on LOTZ and COCZ are from [Laumanns et al., 2004].

Mutation	MOEA	LOTZ	COCZ
One-bit	$\text{MOEA}^{\text{onebit}}$	$\Theta(n^3)$	$O(n^2 \log n)$
	$\text{MOEA}^{\text{onebit}}_{\text{fair}}$	$O(n^2 \log n)$	$\Omega(n^2), O(n^2 \log n)$
	$\text{MOEA}^{\text{onebit}}_{\text{greedy}}$	$O(n^2 \log n)$	$\Theta(n^2)$
	$\text{MOEA}_{\text{recomb},0}^{\text{onebit}}$	$\Theta(n^3)$	$\Omega(n^2), O(n^2 \log n)$
	$\text{MOEA}_{\text{recomb},0.5}^{\text{onebit}}$	$\Theta(n^2)$	$\Theta(n \log n)$
Bit-wise	$\text{MOEA}^{\text{bitwise}}$	$O(n^3)$ [Giel, 2003]	$O(n^2 \log n)$
	$\text{MOEA}_{\text{recomb},0}^{\text{bitwise}}$	$\Omega(n^2), O(n^3)$	$\Omega(n \log n), O(n^2 \log n)$
	$\text{MOEA}_{\text{recomb},0.5}^{\text{bitwise}}$	$\Theta(n^2)$	$\Theta(n \log n)$

Through the analysis, we discover that recombination works in the studied situation by accelerating the filling of the Pareto front through recombining diverse Pareto optimal solutions that have been found. This finding is unique to multi-objective optimization, as there is no Pareto front in single-objective situations.

The rest of this chapter is organized as follows. Section 8.1 introduces some background on theoretically analyzing the recombination operator as well as another common reproduction operator, i.e., mutation. Sections 8.2 and 8.3 introduce $\text{MOEA}_{\text{recomb}}^{\text{onebit}}/\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$, and the running time analysis on Weighted LPTNO and COCZ, respectively. Section 8.4 presents empirical verification. Section 8.5 concludes this chapter.

8.1 Recombination and Mutation

Many theoretical results of EAs have been obtained during the last two decades, e.g., [Neumann and Witt, 2010, Auger and Doerr, 2011]. However, most studies focused on EAs with mutation operators, while only a few included recombination operators.

On the influence of mutation operators, Doerr et al. [2008b] theoretically compared the two common mutation operators, one-bit mutation and bit-wise mutation. The bit-wise mutation operator which searches globally was believed to be at least as good as the one-bit mutation operator which searches locally. By constructing an artificial pseudo-Boolean function, however, Doerr et al. [2008b] showed that bit-wise mutation could be worse than one-bit mutation. The influence of mutation rate $p = c/n$ of bit-wise mutation was then extensively studied. Doerr et al. [2013b] proved that for (1+1)-EA solving the monotonic pseudo-Boolean function class, the expected running time is $\Theta(n \log n)$ and $O(n^{3/2})$ for $c < 1$ and $c = 1$, respectively; when $c \geq 16$, it requires exponential time for some specific monotonic function. Doerr et al. [2017] showed that for (1+1)-EA solving the multimodal problem $\text{Jump}_{m,n}$, a random mutation rate where c is drawn from a power-law distribution can lead to nearly optimal running time for each m . In fact, the dynamic mutation rate has been shown effective with diverse mechanisms, including time-dependent mutation rate [Jansen and Wegener, 2006], rank-based mutation rate [Oliveto et al., 2009b], fitness-dependent mutation rate [Böttcher et al., 2010, Doerr et al., 2013a, Badkobeh et al., 2014] and self-adaptive mutation rate [Dang and Lehre, 2016, Doerr et al., 2018]. The influence of mutation operators has also been theoretically studied for EAs solving some combinatorial optimization problems, e.g., *Eulerian cycle* [Doerr et al., 2006], as well as problems over the domain $\{0, 1, \dots, r\}^n$, where $r > 1$ [Doerr et al., 2006]. Furthermore, Kötzting et al. [2014] proved the importance of mutation in GP.

Next we briefly review theoretical works on recombination operators. In the scenario of single-objective optimization, running time analyses have provided a theoretical understanding of recombination operators. Several recombination-only EAs have been shown to be efficient on the H-IFF problem which is difficult for any kind of mutation-based EAs [Watson, 2001, Dietzfelbinger et al., 2003]. Recombination operators have also been proved to be crucially important on some problems [Jansen and De Jong, 2002, Jansen and Wegener, 2005], and subsequently shown to be useful in more cases such as *Ising* models [Fischer and Wegener, 2005, Sudholt, 2005], the TwoPaths instance class of the problem of computing *unique input-output sequences* [Lehre and Yao, 2008], some instance classes of the vertex cover problem [Oliveto et al., 2008], and the *all-pairs shortest path* problem [Doerr et al., 2008a, Doerr and Theile, 2009, Doerr et al., 2010a]. Meanwhile, on the contrary, Richter et al. [2008] produced the Ignoble Trail functions where a recombination operator has been shown to be harm-

ful. Kötzing et al. [2011b] found that recombination with ideal diversity can lead to drastic speedups on the OneMax and Jump_{*m,n*} problems, and also analyzed how the recombination probability and population size affect population diversity. Recently, the running time analysis of some natural EAs with recombination operators has been performed, e.g., [Oliveto and Witt, 2014, 2015, Sudholt, 2017]. The switch analysis approach presented in Chapter 4 has been used to compare the running time of an EA turning recombination on and off [Yu et al., 2010, 2011]. All these studies are in the scenario of single-objective optimization, and the results are difficult to generalize to multi-objective optimization. In particular, multi-objective optimization aims at finding a set of Pareto optimal solutions rather than a single optimal solution, where the situation becomes more complicated.

Early analyses of MOEAs concentrated on conditions where MOEAs can find the Pareto front given unlimited time, e.g., [Rudolph, 1998, Hanne, 1999, Rudolph and Agapie, 2000]. For running time analyses, studies on two bi-objective pseudo-Boolean problems, i.e., LOTZ and COCZ, have led to some understanding of limited time behaviors of MOEAs [Giel, 2003, Lammans et al., 2004]. Note that none of these previously analyzed MOEAs, as introduced in Section 2.1, uses recombination operators. We rename these MOEAs in Table 8.2 for a clearer presentation. Table 8.1 lists the previous results of these MOEAs for the two problems.

Table 8.2. Unified names of the previously analyzed MOEAs.

Original name	Unified name	Explanation
SEMO	MOEA ^{<i>onebit</i>}	A simple MOEA with one-bit mutation
GSEMO	MOEA ^{<i>bitwise</i>}	A simple MOEA with bit-wise mutation
FEMO	MOEA ^{<i>onebit_fair</i>}	A modification of SEMO where every solution has equal total reproduction chance
GEMO	MOEA ^{<i>onebit_greedy</i>}	A modification of SEMO where only new born solutions dominating some current solutions have reproduction chance

The only work analyzing recombination operators in MOEAs is, to the best of our knowledge, by Neumann and Theile [2010]. They proved that a recombination operator can speed up EAs for the multi-criteria all-pairs shortest path problem. As discovered through their analysis, the recombination operator can be helpful in the interplay with the mutation operator, such that good solutions can be evolved efficiently. It is worth noting that this mechanism is different from what we have found, where the recombination operator works by accelerating the filling of the Pareto front through recombinining diverse Pareto optimal solutions that have been found.

8.2 Recombination Enabled MOEAs

The recombination-incorporated MOEA ($\text{MOEA}_{\text{recomb}}^{\text{onebit}}$) is described in Algorithm 8.1, which is extended from $\text{MOEA}^{\text{onebit}}$ in Algorithm 2.5 by incorporating a recombination operator. The components of $\text{MOEA}_{\text{recomb}}^{\text{onebit}}$ are explained in the following.

Algorithm 8.1 $\text{MOEA}_{\text{recomb}}^{\text{onebit}}$ Algorithm

Input: pseudo-Boolean function vector $\mathbf{f} = (f_1, f_2, \dots, f_m); p_r \in [0, 1]$

Output: set of solutions from $\{0, 1\}^n$

Process:

```

1: let  $P = \text{Initialization}(m \text{ solutions randomly from } \{0, 1\}^n)$ ;
2: while criterion is not met do
3:   for each of the randomly selected  $m/2$  objectives, select the best solution
      from  $P$  to form  $P_s^1$ ;
4:   select  $(m - |P_s^1|)$  solutions randomly from  $P \setminus P_s^1$  to form  $P_s^2$ ;
5:    $P_s = P_s^1 \cup P_s^2$ ;
6:    $r =$  a value chosen from  $[0, 1]$  uniformly at random;
7:   if  $r < p_r$  then
8:     apply recombination on  $P_s$  to generate  $P_o$ 
9:   else
10:    apply one-bit mutation on each solution  $s \in P_s$  to generate  $P_o$ 
11:   end if
12:   for each solution  $s' \in P_o$ 
13:     if  $\nexists z \in P$  such that  $z \succeq s'$  then
14:        $P = (P \setminus \{z \in P \mid s' \succ z\}) \cup \{s'\}$ 
15:     end if
16:   end for
17: end while
18: return  $P$ 

```

It is well known that the diversity of a population is important to the success of recombination, as recombination makes no progress from similar solutions. To employ diversity control in $\text{MOEA}_{\text{recomb}}^{\text{onebit}}$, we use an *objective diversity* measure based on the assumption that the diversity of a set of solutions is consistent with the difference between their objective vectors. By this assumption, a population has a high diversity if it contains solutions with good objective values on different objectives. Thus, we define the objective diversity of a set of solutions as the number of objectives, on which at least one solution in this set has a good objective value. Formally, for a set P of solutions, define a variable q_i for the i -th objective ($1 \leq i \leq m$),

$$q_i = \begin{cases} 1 & \text{if } \max\{f_i(s) \mid s \in P\} \geq \theta_i; \\ 0 & \text{otherwise,} \end{cases} \quad (8.1)$$

where θ_i is the “goodness” threshold of the i -th objective, then the objective diversity of P is $\sum_{i=1}^m q_i$. Given m objectives, the largest value of objective diversity is m . Here, we use the objective diversity with θ_i equal to the minimum local optimal value of the i -th objective, and a local optimal solution is with respect to the neighbor space containing all solutions having Hamming distance 1 from the current solution.

To make the initial population diverse enough, we use an initialization process in Definition 8.1, where m independent runs of RLS are employed to optimize the m objective functions f_1, f_2, \dots, f_m , and each RLS corresponds to one objective. As introduced in Section 2.1, RLS is Algorithm 2.1 using one-bit mutation rather than bit-wise mutation. In RLS, the solution is examined to see whether it is local optimal for an objective in every n mutation steps, where n is the length of a solution. This initialization process is terminated when local optimal solutions have been found for all objectives.

Definition 8.1 (Initialization). *Input: solutions s_1, s_2, \dots, s_m from $\{0, 1\}^n$; Output: m local optimal solutions each corresponding to one of the m objectives; Process:*

1. **repeat**
2. **repeat** the following process n times
3. Create s'_1, s'_2, \dots, s'_m by flipping a randomly chosen bit of s_1, s_2, \dots, s_m , respectively;
4. **if** $f_i(s'_i) \geq f_i(s_i)$ for an i **then**
5. $s_i = s'_i$
6. **end if**
7. **end repeat**
8. **until** $\forall i : s_i$ is a local optimal solution for f_i

This process makes the initial population contain one good solution for each objective, i.e., $\forall i \in [m] : q_i = 1$. Thus, the objective diversity of the initial population is m . A solution in the population will be removed only if it is dominated by a new solution, so there always exist good solutions for each objective, making the objective diversity of the population keep as m , the maximum objective diversity, throughout the evolutionary process.

In each reproduction step, MOEA_{recomb}^{onebit} picks a set of m solutions with objective diversity of at least $m/2$ from the current population to carry out recombination. For this purpose, the best solutions each for one of the randomly selected $m/2$ objectives are selected at first, denoted as a set P_s^1 . The remaining $(m - |P_s^1|)$ solutions are chosen randomly from the population excluding P_s^1 . Thus, the selected solutions for recombination have an objective diversity of at least $m/2$.

In reproduction, a parameter p_r is used to control the use of recombination. In other words, in each reproduction step, the offspring solutions are generated by recombination with probability p_r , otherwise generated by mutation. MOEA_{recomb,0.5}^{onebit} and MOEA_{recomb,0}^{onebit} correspond to MOEA_{recomb}^{onebit} with $p_r = 0.5$ and $p_r = 0$, respectively. At the end of each iteration, the

offspring solutions P_o are used to update the current population. For an offspring solution, if it is not weakly dominated by any solution in the current population, it will be included and then the population is updated by removing all solutions that are no longer non-dominated.

The recombination operator employed in MOEA_{recomb}^{onebit} is the diagonal multi-parent recombination operator [Eiben et al., 1994] in Definition 8.2. For m solutions, diagonal multi-parent recombination selects $(m - 1)$ recombination points between adjacent bits randomly and creates m offspring solutions by combining the components partitioned by the recombination points sequentially. Figure 8.1 gives an example of diagonal multi-parent recombination over three Boolean vector solutions. This recombination operator is a generalization of one-point recombination over two solutions shown in Figure 1.5.

Definition 8.2 (Recombination [Eiben et al., 1994]). *Given m solutions of length n , select $(m - 1)$ recombination points randomly from $(n - 1)$ positions between adjacent bits, and create m offspring solutions as follows. Denote the order of the m parent solutions as $1, 2, \dots, m$. The m offspring solutions are generated by combining m components partitioned by the $(m - 1)$ recombination points, where the components of the i -th ($1 \leq i \leq m$) offspring solution come sequentially from parents $i, i + 1, \dots, m - 1, m, 1, \dots, i - 1$.*

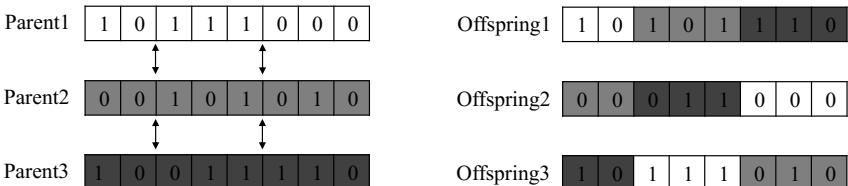


Figure 8.1. Diagonal multi-parent recombination [Eiben et al., 1994] over three Boolean vector solutions.

In the algorithm MOEA_{recomb}^{onebit}, the employed mutation operator is the one-bit mutation operator, i.e., line 10 of Algorithm 8.1, which flips a randomly chosen bit of a solution. If there are a set of non-dominated solutions such that they weakly dominate all their Hamming neighbors, and the current population is contained in this set, the population cannot escape from this set through the one-bit mutation operator. To address this issue, we can modify the one-bit mutation operator in MOEA_{recomb}^{onebit} to a more general mutation operator, i.e., the bit-wise mutation operator, which flips each bit of a solution with probability $1/n$. One-bit mutation searches locally, whereas bit-wise mutation searches globally. The modified algorithm is called MOEA_{recomb}^{bitwise}, which is almost the same as MOEA_{recomb}^{onebit}. The only difference is that line 10 of MOEA_{recomb}^{bitwise} is “apply bit-wise mutation on each

solution $s \in P_s$ to generate P_o ” whereas that of MOEA_{recomb}^{onebit} is “apply one-bit mutation on each solution $s \in P_s$ to generate P_o ”. As MOEA_{recomb}^{onebit} is an extension of MOEA^{onebit} by incorporating recombination, MOEA_{recomb}^{bitwise} can be viewed as an extension of MOEA^{bitwise} using the same way.

8.3 Case Study

In this section, we will use Weighted LPTNO and COCZ to study the influence of recombination operators for MOEAs. Weighted LPTNO in Definition 8.3 is a bi-objective problem class, which is generalized from the LOTZ problem in Definition 2.7. The first objective is to maximize the number of leading positive 1-bits, and the other objective is to maximize the number of trailing negative 1-bits.

Definition 8.3 (Weighted LPTNO). *The Weighted LPTNO problem of size n is to find solutions from $\{-1, 1\}^n$ which maximize*

$$\mathbf{f}(s) = \left(\sum_{i=1}^n w_i \prod_{j=1}^i (1 + s_j), \sum_{i=1}^n v_i \prod_{j=i}^n (1 - s_j) \right), \quad (8.2)$$

where s_j denotes the j -th element of $s \in \{-1, 1\}^n$, and $\forall i \in [n] : w_i, v_i > 0$.

The objective vector of Weighted LPTNO can be represented as

$$\left(\sum_{k=1}^i 2^k w_k, \sum_{k=n-j+1}^n 2^{n+1-k} v_k \right), \quad (8.3)$$

where i and j are the number of leading positive 1-bits and trailing negative 1-bits, respectively. The Pareto front is $\{(\sum_{k=1}^i 2^k w_k, \sum_{k=i+1}^n 2^{n+1-k} v_k) \mid 0 \leq i \leq n\}$, and the corresponding Pareto optimal solutions are $(-1)^n, 1(-1)^{n-1}, \dots, 1^{n-1}(-1), 1^n$.

It is worth noting that Weighted LPTNO, though can be considered as an extension of LOTZ by shifting the solution space from $\{0, 1\}^n$ to $\{-1, 1\}^n$ and adding weights w_i and v_i , has very different properties from LOTZ. For LOTZ, the Pareto front is symmetric, i.e., if (a, b) is in the Pareto front then (b, a) is also in the Pareto front. For Weighted LPTNO, however, the Pareto front can be non-symmetric in general, leading to much more complicated situations. Note that for solving Weighted LPTNO by MOEAs, the flipping of one bit means that changing the bit from “-1” to “1” or from “1” to “-1”.

The COCZ problem in Definition 2.8 is to maximize two linear functions over $\{0, 1\}^n$. The first objective is to maximize the number of 1-bits, and the other objective is to maximize the number of 1-bits in the first half of the solution plus the number of 0-bits in the second half. The Pareto front is $\{(n/2, n), (n/2 + 1, n - 1), \dots, (n, n/2)\}$, and the set of all Pareto optimal solutions are $\{1^{\frac{n}{2}} * 1^{\frac{n}{2}} \mid * \in \{0, 1\}\}$, with the size of $2^{\frac{n}{2}}$.

8.3.1 Weighted LPTNO

We consider the running time of the initialization process of MOEA_{recomb}^{onebit} and MOEA_{recomb}^{bitwise}, which is to optimize the two objectives of a problem separately by two independent RLS. For Weighted LPTNO, the objectives have the same structure as the LeadingOnes problem in Definition 2.4, for which RLS takes $\Theta(n^2)$ time to optimize by Theorem 13 in [Johannsen et al., 2010].

During the optimization of RLS in the initialization process, a solution is examined whether it is local optimal for an objective every n mutation steps. The running time of examining once is n because the neighborhood size of a solution is n . Thus, the total running time of examination is the same as that of optimization. We then have the following lemma.

Lemma 8.1. *On Weighted LPTNO, the expected running time of the initialization procedure of MOEA_{recomb}^{onebit}/MOEA_{recomb}^{bitwise} is $\Theta(n^2)$.*

Next, we focus on the population size during the evolutionary process.

Lemma 8.2. *On Weighted LPTNO, the population size of MOEAs maintaining non-dominated solutions is always no larger than $n + 1$, and is equal to $(n + 1)$ iff the population consists of all Pareto optimal solutions.*

Proof. Because the solutions in the population P have a one-to-one correspondence with their objective vectors $f(P) = \{f(s) \mid s \in P\}$, we only need to analyze the size of $f(P)$.

The first objective of Weighted LPTNO has $(n + 1)$ possible values, i.e., $\{\sum_{i=1}^j 2^i w_i \mid 0 \leq j \leq n\}$. In the population P , any possible value of the first objective can have at most one corresponding value of the second objective, because two solutions with the same first objective value are comparable, violating the property that the solutions in the population are non-dominated. Thus, the size of $f(P)$ is no larger than $n+1$. If the size of $f(P)$ is $n+1$, the values on the first dimension of the $(n+1)$ objective vectors in $f(P)$ are $0, \sum_{i=1}^1 2^i w_i, \dots, \sum_{i=1}^n 2^i w_i$, respectively. Because the solutions in the population are non-dominated, the corresponding values on the second dimension must decrease, i.e., they are $\sum_{i=1}^n 2^{n+1-i} v_i, \sum_{i=2}^n 2^{n+1-i} v_i, \dots, 0$, respectively. Thus, $f(P)$ is the Pareto front. \square

In the evolutionary process of the MOEAs analyzed in this chapter, the population contains at most one Pareto optimal solution for each objective vector in the Pareto front, and the Pareto optimal solution will never be eliminated once found. Thus, to find the Pareto front, it is sufficient to increase the number of Pareto optimal solutions enough times, e.g., the size of the Pareto front. We call it a *success* if the number of Pareto optimal solutions increases by one step. By analyzing the expected steps for success and summing up the expected steps for all required successes, we can get the expected running time bound of the MOEAs. In the following proof, this idea will often be used.

Note that bi-objective problems are studied, and thus, when solutions for reproduction in $\text{MOEA}_{\text{recomb}}^{\text{onebit}}$ and $\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ are to be selected, the best solution in the current population for a randomly selected objective is picked up at first, and then the other solution is chosen randomly from the remaining solutions. Also, because bi-objective problems are considered, the recombination operator used is simply one-point recombination.

Theorem 8.1. *The expected running time of $\text{MOEA}_{\text{recomb},0.5}^{\text{onebit}}/\text{MOEA}_{\text{recomb},0.5}^{\text{bitwise}}$ on the Weighted LPTNO problem is $\Theta(n^2)$.*

Proof. For Weighted LPTNO, the size of the Pareto front is $n + 1$. After initialization, the two Pareto optimal solutions 1^n and $(-1)^n$ have been found. Thus, it is sufficient to increase the number of Pareto optimal solutions for $(n - 1)$ times to find the Pareto front.

We consider the expected steps for a success when starting from a population P containing $(i + 1)$ Pareto optimal solutions, where $i \in [n - 1]$. In the selection procedure, because the two solutions 1^n and $(-1)^n$, which are optimal for the two objectives of Weighted LPTNO, respectively, have been found, it actually selects one solution s randomly from $\{1^n, (-1)^n\}$ at first, and then selects the other solution randomly from the remaining solutions $P \setminus \{s\}$. Consider two cases in selection for the probability q of generating one new Pareto optimal solution by one-point recombination in one step: the two solutions 1^n and $(-1)^n$ are selected; one selected solution is either 1^n or $(-1)^n$ and the other selected one is a Pareto optimal solution from $P \setminus \{1^n, (-1)^n\}$. In the former case occurring with probability $1/(|P| - 1)$, $q = (n - i)/(n - 1)$. In the latter case occurring with probability $1/(2(|P| - 1))$, suppose that the Pareto optimal solution selected from $P \setminus \{1^n, (-1)^n\}$ has k ($0 < k < n$) leading positive 1-bits, the number of Pareto optimal solutions having more than k leading positive 1-bits in the current population is k' ($1 \leq k' \leq i - 1$) and the other selected solution is 1^n , then $q = (n - k - k')/(n - 1)$. Correspondingly, when the other selected solution is $(-1)^n$, $q = (k - i + k')/(n - 1)$. Thus, in the latter case, $q = (n - k - k')/(n - 1) + (k - i + k')/(n - 1) = (n - i)/(n - 1)$. By combining these two cases, we have

$$q = \frac{1}{|P| - 1} \cdot \frac{n - i}{n - 1} + \frac{i - 1}{2(|P| - 1)} \cdot \frac{n - i}{n - 1} = \frac{(i + 1)(n - i)}{2(|P| - 1)(n - 1)}, \quad (8.4)$$

where the factor $(i - 1)$ is included because there are $(i - 1)$ Pareto optimal solutions in $P \setminus \{1^n, (-1)^n\}$ for selection in the latter case. Because the recombination probability is $1/2$, the number of Pareto optimal solutions increases by one step with probability at least $(i+1)(n-i)/(4(|P|-1)(n-1)) \geq (i+1)(n-i)/(4(n-1)^2)$, where the inequality holds by $|P| \leq n$ from Lemma 8.2. Thus, the expected number of steps for a success when starting from a population containing $(i + 1)$ Pareto optimal solutions is at most $4(n - 1)^2 / ((i + 1)(n - i))$.

Because two offspring solutions need to be evaluated in one reproduction step, the running time of one step is counted as 2. Thus, the expected running time to find the Pareto front after initialization is at most $2 \cdot \sum_{i=1}^{n-1} 4(n-1)^2 / ((i+1)(n-i)) = O(n \log n)$. By combining the expected running time $\Theta(n^2)$ of initialization in Lemma 8.1, the expected running time of $\text{MOEA}_{\text{recomb},0.5}^{\text{onebit}} / \text{MOEA}_{\text{recomb},0.5}^{\text{bitwise}}$ on Weighted LPTNO is $\Theta(n^2)$. \square

Thus, we have proved that the expected running time of $\text{MOEA}_{\text{recomb}}^{\text{onebit}}$ or $\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ with the recombination probability 0.5 on Weighted LPTNO is $\Theta(n^2)$. The running time on the LOTZ problem, a special case of Weighted LPTNO, has been well studied for the MOEAs with mutation only. Compared with the previously analyzed MOEAs that are facilitated with one-bit mutation or bit-wise mutation, as shown in the third column of Table 8.1, $\text{MOEA}_{\text{recomb},0.5}^{\text{onebit}}$ or $\text{MOEA}_{\text{recomb},0.5}^{\text{bitwise}}$ has better running time on LOTZ.

Next, we analyze the running time of $\text{MOEA}_{\text{recomb}}^{\text{onebit}}$ and $\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ turning off recombination on Weighted LPTNO to see whether recombination is crucial for the efficiency of $\text{MOEA}_{\text{recomb}}^{\text{onebit}}$ and $\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$.

Theorem 8.2. *The expected running time of $\text{MOEA}_{\text{recomb},0}^{\text{onebit}}$ on the Weighted LPTNO problem is $\Theta(n^3)$.*

Proof. For Weighted LPTNO, the offspring Pareto optimal solution generated by one-bit mutation on a Pareto optimal solution must be adjacent to the parent one. After initialization, the two Pareto optimal solutions 1^n and $(-1)^n$ have been found. Thus, the population in the evolutionary process is always constructed by two continuous subsets L and R of the set $\{1^n, 1^{n-1}(-1), \dots, (-1)^n\}$, where $1^n \in L$ and $(-1)^n \in R$. We divide the optimization process into n phases, where the population in the i -th phase ($1 \leq i \leq n$) consists of $(i+1)$ Pareto optimal solutions and the n -th phase corresponds to the phase where the Pareto front has been found. In the following, we will consider the probability of generating new Pareto optimal solutions by one step in each phase.

In the first phase, $|L| = 1$ and $|R| = 1$. For reproduction, the two solutions 1^n and $(-1)^n$ are selected. For 1^n , the offspring solution generated by one-bit mutation can be accepted only if the rightmost positive 1-bit is mutated. For $(-1)^n$, the offspring solution can be accepted only if the leftmost negative 1-bit is mutated. Thus, by one step, two new Pareto optimal solutions will be generated simultaneously with probability $1/n^2$; only one new Pareto optimal solution will be generated with probability $2(n-1)/n^2$; otherwise, no new Pareto optimal solution will be generated.

Next, consider the i -th phase, where $1 < i \leq n-1$. In the selection procedure, because 1^n and $(-1)^n$, which are optimal for the two objectives of Weighted LPTNO, respectively, have been found, one solution will be selected randomly from $\{1^n, (-1)^n\}$, and the other solution will be selected randomly from the remaining solutions. The probabilities for two solutions selected by this procedure are $1/2$ and $1/i$, respectively. There are two cases.

Case 1: $\min\{|L|, |R|\} > 1$. In this case, one-bit mutation on either 1^n or $(-1)^n$ will never generate new Pareto optimal solutions, and only the rightmost solution of L or the leftmost solution of R can generate one new Pareto optimal solution with probability $1/n$ by one-bit mutation. Thus, one new Pareto optimal solution can be generated by one step with probability $2/(in)$; otherwise, no new Pareto optimal solution will be generated.

Case 2: $\min\{|L|, |R|\} = 1$. Suppose $|L| = 1$. If the solution selected from $\{1^n, (-1)^n\}$ is $(-1)^n$, one-bit mutation on $(-1)^n$ will never generate new Pareto optimal solutions, and only when the other selected solution is 1^n or the leftmost solution of R , mutation can generate one new Pareto optimal solution from it with probability $1/n$. If the solution selected from $\{1^n, (-1)^n\}$ is 1^n , by mutation on 1^n , one new Pareto optimal solution $1^{n-1}(-1)$ can be generated with probability $1/n$, and only when the other selected solution is the leftmost solution of R , one new Pareto optimal solution can be generated by mutation with probability $1/n$. Thus, when $i < n - 1$, by one step, only one new Pareto optimal solution will be generated while $\min\{|L|, |R|\}$ remains 1 with probability $1/(in) - 1/(2in^2)$; one or two new Pareto optimal solutions will be generated while $\min\{|L|, |R|\} > 1$ with probability $1/(2n) + 1/(2in)$; otherwise, no new Pareto optimal solution will be generated. When $i = n - 1$, the last undiscovered Pareto optimal solution will be found in one step with probability $(n^2 + 2n - 1)/(2(n - 1)n^2)$.

During the evolutionary process after initialization, the state of the population will change as follows:

1. start at the 1st phase;
2. transfer to case 2;
3. stay at case 2;
4. transfer to case 1;
5. stay at case 1;
6. end at the n -th phase, i.e, the Pareto front is found.

Steps 2 and 3 may be skipped, because the state of the population can transfer directly to case 1 when leaving the 1st phase. Steps 4 and 5 may also be skipped, because the state of the population can always stay at case 2 until the Pareto front is found.

According to the analysis above, the probability of generating one new Pareto optimal solution at steps 3 and 5 is $\Theta(1/(in))$; the probability of transferring at steps 2 and 4 is $\Omega(1/n^2)$. Thus, the expected number of steps after initialization to generate the Pareto front is $\Theta(\sum_{i=1}^{n-1} in) = \Theta(n^3)$. By combining the expected running time $\Theta(n^2)$ of initialization in Lemma 8.1, the total expected running time is $\Theta(n^3)$. \square

Theorem 8.3. *The expected running time of MOEA_{recomb,0}^{bitwise} on the Weighted LPTNO problem is $\Omega(n^2)$ and $O(n^3)$.*

Proof. After the initialization procedure, the two Pareto optimal solutions 1^n and $(-1)^n$ have been found. Thus, it is sufficient to increase the number of Pareto optimal solutions for $(n - 1)$ times to find the Pareto front.

Before finding the Pareto front, there always exists at least one Pareto optimal solution in the current population, from which one new Pareto optimal solution can be generated by flipping only the rightmost positive 1-bit or the leftmost negative 1-bit. Because the probability of selecting a specific solution is at least $1/(n - 1)$ by Lemma 8.2, the number of Pareto optimal solutions increases by one step with probability at least $(1/(n - 1)) \cdot (1/n)(1 - 1/n)^{n-1} \geq 1/(en(n - 1))$. Thus, the expected running time to find the Pareto front after initialization is at most $2en(n - 1)^2$. Combining the expected running time $\Theta(n^2)$ of initialization in Lemma 8.1, the total expected running time is $\Omega(n^2)$ and $O(n^3)$. \square

The above two theorems show that without recombination, the expected running time of $\text{MOEA}_{\text{recomb}}^{\text{onebit}}/\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ on the Weighted LPTNO problem is $\Theta(n^3)/\Omega(n^2)$, increasing from $\Theta(n^2)$ of $\text{MOEA}_{\text{recomb}}^{\text{onebit}}/\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ with the recombination probability 0.5. Thus, recombination is crucial for the efficiency of $\text{MOEA}_{\text{recomb}}^{\text{onebit}}/\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ on Weighted LPTNO.

From the analysis of $\text{MOEA}_{\text{recomb}}^{\text{onebit}}$ and $\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ on Weighted LPTNO, we can find that recombination works by accelerating the filling of the Pareto front through recombining diverse Pareto optimal solutions found-so-far. For example, when the two diverse Pareto optimal solutions 1^n and $1^{n/2}(-1)^{n/2}$ are selected for reproduction, the probability of generating offspring Pareto optimal solutions that are different from the parents by one-point recombination is $(n/2 - 1)/(n - 1)$, whereas the probability by mutation (one-bit or bit-wise) on any two solutions is at most $4/n$.

8.3.2 COCZ

The initialization process of $\text{MOEA}_{\text{recomb}}^{\text{onebit}}$ and $\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ is to optimize the two objectives of COCZ, which have the same structure as the OneMax problem in Definition 2.3, by two independent RLS. By Theorem 11 in [Johannsen et al., 2010], the running time of RLS on OneMax is $\Theta(n \log n)$. During the optimization of RLS in the initialization process, a solution is examined if it is local optimal for an objective every n mutation steps. The running time of examining once is n . Thus, the total running time of examination is the same as that of optimization, leading to the following lemma.

Lemma 8.3. *On COCZ, the expected running time of the initialization procedure of $\text{MOEA}_{\text{recomb}}^{\text{onebit}}/\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ is $\Theta(n \log n)$.*

Lemma 8.4 shows an upper bound of the population size. The proofs of the following lemma and theorems are provided in Appendix A.3.

Lemma 8.4. *On COCZ, the population size of MOEAs maintaining non-dominated solutions is always no larger than $n/2 + 1$.*

Theorem 8.4. *The expected running time of $\text{MOEA}_{\text{recomb},0.5}^{\text{onebit}}/\text{MOEA}_{\text{recomb},0.5}^{\text{bitwise}}$ on the COCZ problem is $\Theta(n \log n)$.*

Theorem 8.4 shows that the expected running time of $\text{MOEA}_{\text{recomb}}^{\text{onebit}}$ or $\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ with the recombination probability 0.5 on COCZ is $\Theta(n \log n)$. Compared with the previously analyzed MOEAs with one-bit mutation, as shown in the last cell of the second row of Table 8.1, $\text{MOEA}_{\text{recomb},0.5}^{\text{onebit}}$ has better running time on COCZ. For $\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$, it uses bit-wise mutation. Note that $\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ is the only previously analyzed MOEA with bit-wise mutation, and its expected running time on COCZ is unknown. Thus, for the comparison purpose, we derive the running time of $\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ on COCZ.

Theorem 8.5. *The expected running time of $\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ on the COCZ problem is $O(n^2 \log n)$.*

Compared with $\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ on COCZ, $\text{MOEA}_{\text{recomb},0.5}^{\text{bitwise}}$ has a better upper bound of running time. Next, we analyze the running time of $\text{MOEA}_{\text{recomb}}^{\text{onebit}}$ and $\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ turning off recombination on COCZ to see whether recombination is crucial for their efficiency.

Theorem 8.6. *The expected running time of $\text{MOEA}_{\text{recomb},0}^{\text{onebit}}$ on the COCZ problem is $\Omega(n^2)$ and $O(n^2 \log n)$.*

Theorem 8.7. *The expected running time of $\text{MOEA}_{\text{recomb},0}^{\text{bitwise}}$ on the COCZ problem is $\Omega(n \log n)$ and $O(n^2 \log n)$.*

The above two theorems show that without recombination, the expected running time of $\text{MOEA}_{\text{recomb}}^{\text{onebit}}/\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ on COCZ is $\Omega(n^2)/\Omega(n \log n)$, increasing from $\Theta(n \log n)$ of $\text{MOEA}_{\text{recomb}}^{\text{onebit}}/\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ with the recombination probability 0.5. Thus, recombination is crucial for the efficiency of $\text{MOEA}_{\text{recomb}}^{\text{onebit}}/\text{MOEA}_{\text{recomb}}^{\text{bitwise}}$ on the COCZ problem. From the analysis, we can also find that recombination works by accelerating the filling of the Pareto front through recombining diverse Pareto optimal solutions found-so-far, as that has been found from the analysis on Weighted LPTNO.

8.4 Empirical Verification

In this section, we conduct experiments to complement the theoretical analysis, because some running time bounds in Table 8.1 are not tight, making the comparison of performance between MOEAs not strict. For example, when comparing $\text{MOEA}_{\text{recomb},0.5}^{\text{onebit}}$ with $\text{MOEA}_{\text{recomb}}^{\text{onebit}}$ on COCZ, one can only say that the expected running time of $\text{MOEA}_{\text{recomb},0.5}^{\text{onebit}}$ is better than the upper bound of the expected running time of $\text{MOEA}_{\text{recomb}}^{\text{onebit}}$ proved-so-far. To complement the comparison, we estimate the running time order by experiments. On each problem size, we repeat independent runs of an MOEA for

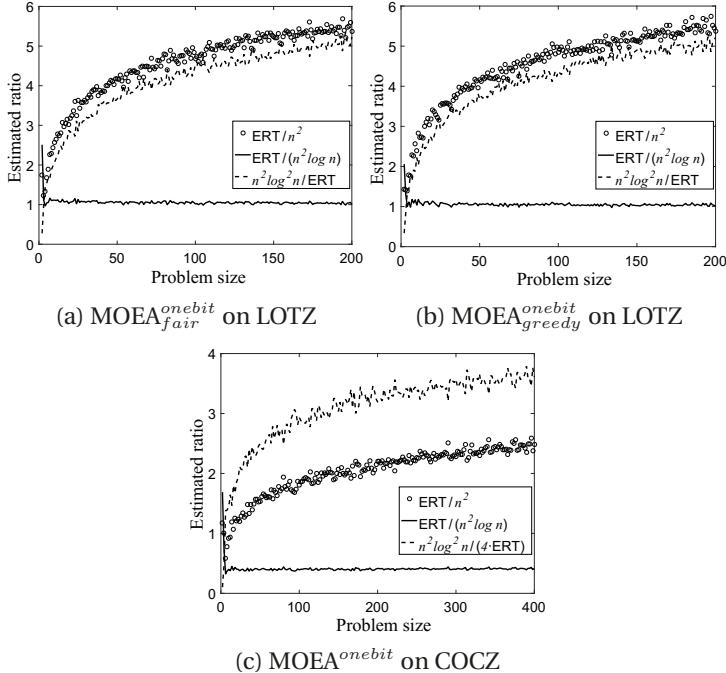


Figure 8.2. Estimated expected running time (ERT) of several MOEAs with one-bit mutation on LOTZ and COCZ. Note that the base of the logarithm is e .

1,000 times, and the average running time is recorded as an estimation of the expected running time. Figures 8.2 and 8.3 plot the results.

From Figure 8.2(a), we can observe that for MOEA_{fair}^{onebit} on LOTZ, the curve of the expected running time divided by $n^2 \log n$ tends to be a constant, and both the curve of the expected running time divided by n^2 and the curve of $n^2 \log^2 n$ divided by the expected running time grow in a closely logarithmic trend. Thus, the observation suggests that the expected running time of MOEA_{fair}^{onebit} is approximately in the order of $n^2 \log n$. Similarly, Figures 8.2(b-c) suggest that both the expected running time of MOEA_{greedy}^{onebit} on LOTZ and the expected running time of MOEA^{onebit} on COCZ is approximately in the order of $n^2 \log n$. Thus, by combining the proved results, we can conclude that MOEA_{recomb,0.5}^{onebit} is the most efficient algorithm among the MOEAs with one-bit mutation on LOTZ and COCZ.

Figure 8.3 suggests that the expected running time of MOEA^{bitwise} and MOEA_{recomb,0}^{bitwise} on LOTZ is approximately in the order of n^3 , and the expected running time of MOEA^{bitwise} and MOEA_{recomb,0}^{bitwise} on COCZ is approximately in the order of $n^2 \log n$ and $n^2 / \log n$, respectively. Therefore, we can conclude that MOEA_{recomb,0.5}^{bitwise} is the most efficient algorithm among the MOEAs with bit-wise mutation on these two problems.

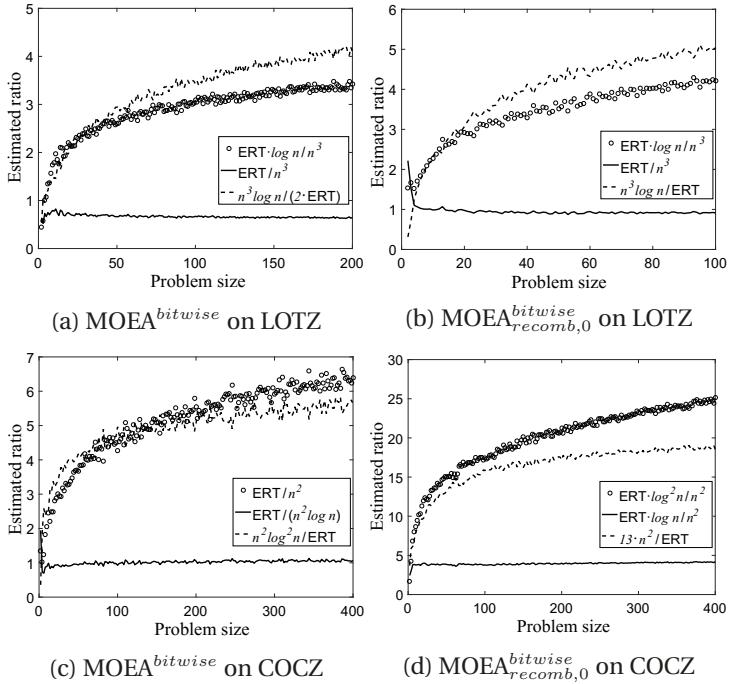


Figure 8.3. Estimated expected running time (ERT) of several MOEAs with bit-wise mutation on LOTZ and COCZ. Note that the base of the logarithm is e .

8.5 Summary

In this chapter, we theoretically study the influence of recombination operators in multi-objective evolutionary optimization, through analyzing the running time of two MOEAs with a recombination operator: MOEA^{*onebit*}_{*recomb*} using one-bit mutation and MOEA^{*bitwise*}_{*recomb*} using bit-wise mutation, on two model problems, i.e., Weighted LPTNO and COCZ. The results of MOEA^{*onebit*}_{*recomb*} and MOEA^{*bitwise*}_{*recomb*} turning recombination on and off on these two problems show the helpfulness of recombination. By comparing with the previously analyzed MOEAs on the LOTZ (a special case of Weighted LPTNO) and COCZ problems, we find that MOEA^{*onebit*} and MOEA^{*bitwise*}_{*recomb*} are the most efficient. This supports the conclusion that recombination operators can be useful for multi-objective evolutionary optimization.

The analysis discloses that the recombination operator works in the studied situation by accelerating the filling of the Pareto front through recombining diverse Pareto optimal solutions found-so-far. This finding, though proved only in the studied cases, is likely to hold in general situations and can be enlightening for designing more efficient MOEAs.



Representation

In Section 1.2, we start from a simple solution representation of subsets, i.e., a subset is represented as a binary vector of membership indication from a universe set. Meanwhile, there can be different ways of solution representation, even for the same problem. For example, while GA [Goldberg, 1989] usually represents solutions by binary vectors on which bit-wise mutation and recombination operators are applied to reproduce new solutions, GP [Koza, 1994] usually represents solutions using tree structures on which node-wise mutation and recombination operators are applied. Note that different ways of representation lead to different solution spaces, and can have a significant influence on the performance of EAs.

This chapter theoretically studies the influence of solution representation on the performance of EAs. The importance of solution representation in evolutionary optimization has been empirically studied before [Rothlauf, 2006, Hoai et al., 2006]. For problems where solutions have complex structures, such as in evolving computer programs, electronic design and symbolic regression, structured representation of solutions is more natural and efficient than vector representation. This intuition has got support by empirical studies, showing that GP and *ant colony optimization* (ACO) with variable-structured representation are able to produce better optimization performance [Poli et al., 2008, Koza, 2010, Khan et al., 2014]. Unfortunately, such intuition has received little theoretical justification [Poli et al., 2010]. In this chapter, we present the running time comparison [Qian et al., 2015d] between GA with a binary-vector representation and GP with a tree-structured representation on two representative combinatorial optimization problems with structured solutions, i.e., the *maximum matching* and *minimum spanning tree* problems.

Previous running time analysis of GP was carried out mainly on simple synthetic problems, due to the complicated behaviors of GP. The (1+1)-GP algorithm, a simple GP which employs population size 1, has been shown to be able to solve some simple synthetic problems in polynomial running time, including the Order and Majority problems [Durrett et al., 2011], the

Sorting problem [Wagner and Neumann, 2014], the Max problem [Kötzing et al., 2014] and the weighted Order problem [Nguyen et al., 2013]. Kötzing et al. [2011a] analyzed GP under the probably approximately correct (PAC) learning framework. The performance of GP has also been analyzed for the problem of evolving Boolean conjunctions [Lissovoi and Oliveto, 2018]. Recently, a simple multi-objective GP, SMO-GP, has been analyzed on some bi-objective optimization problems, transformed from the analyzed single-objective problems, i.e., Order, Majority, and Sorting [Neumann, 2012, Wagner and Neumann, 2012, Nguyen et al., 2013].

The running time of GA has been analyzed on the maximum matching and minimum spanning tree problems. For maximum matchings, (1+1)-EA obtains an $(1/(1 + \epsilon))$ -approximation ratio in $O(m^2\lceil\epsilon^{-1}\rceil)$ expected running time [Giel and Wegener, 2003], where m is the number of edges of the given graph. For minimum spanning trees, (1+1)-EA needs $O(m^2(\log n + \log w_{\max}))$ expected time for finding an optimal solution [Neumann and Wegener, 2007], where m , n and w_{\max} are the number of edges, the number of nodes and the maximum edge weight, respectively. Later, Doerr et al. [2012c] gave a coefficient for this upper bound, i.e., $2em^2(1 + \log m + \log w_{\max})$. By formulating the minimum spanning tree problem as a bi-objective optimization problem, GSEMO, i.e., Algorithm 2.5 using bit-wise mutation, can solve it in $O(mn(n + \log w_{\max}))$ expected time [Neumann and Wegener, 2006], better than (1+1)-EA on dense graphs, e.g., $m = \Theta(n^2)$.

This chapter presents a running time analysis of GP on maximum matchings and minimum spanning trees. We prove that the expected running time of (1+1)-GP on maximum matchings is $O((3m \min\{m, n\}/2)^{\lceil\epsilon^{-1}\rceil})$ for obtaining an $(1/(1 + \epsilon))$ -approximation ratio, and that on minimum spanning trees is $O(mn(\log n + \log w_{\max}))$ for finding an optimal solution, better than the corresponding best running time bounds of (1+1)-EA proved-so-far shown in the second row of Table 9.1, because the number n of nodes is usually smaller than the number m of edges. We also prove that the expected running time of SMO-GP on minimum spanning trees is $O(mn^2)$, better than the running time bound $O(mn(n + \log w_{\max}))$ of GSEMO. Note that we do not make the comparison for SMO-GP and GSEMO on maximum matchings, as the expected running time of GSEMO on maximum matchings has not been known. Our results provide theoretical evidence showing the effectiveness of rich representation in evolutionary optimization. These findings are also verified by experiments. From the analysis, we find that the variable solution structure can be helpful in evolutionary optimization if the solution complexity can be well controlled.

The rest of this chapter is organized as follows. Section 9.1 introduces (1+1)-GP and SMO-GP. The running time analyses on the maximum matching and minimum spanning tree problems are presented in Sections 9.2 and 9.3, respectively. Section 9.4 presents empirical verification. Section 9.5 concludes this chapter.

Table 9.1. Expected running time comparison of GA and GP on the maximum matching and minimum spanning tree problems, where the running time bounds of GA are from [Giel and Wegener, 2003, Neumann and Wegener, 2006, 2007, Doerr et al., 2012c].

	Maximum matching	Minimum spanning tree
GA: (1+1)-EA	$O(m^{2\lceil \epsilon^{-1} \rceil})$	$O(m^2(\log n + \log w_{\max}))$, $\leq 2em^2(1 + \log m + \log w_{\max})$
GP: (1+1)-GP	$O((3m \min\{m, n\}/2)^{\lceil \epsilon^{-1} \rceil})$	$O(mn(\log n + \log w_{\max}))$
GA: GSEMO	/	$O(mn(n + \log w_{\max}))$
GP: SMO-GP	/	$O(mn^2)$

9.1 Genetic Programming Representation

Solutions in GP are usually represented by tree structures. Given a set F of functions, e.g., arithmetic operators, and a set T of terminals, e.g., variables, an internal node of a tree denotes a function in F and a leaf node denotes a terminal in T .

(1+1)-GP in Algorithm 9.1 is a simple GP algorithm [Durrett et al., 2011]. It first initializes a solution, e.g., by generating a random tree, and then repeatedly generates an offspring solution by mutation and updates the parent solution if the offspring solution is not worse. Note that maximization problems are considered in Algorithm 9.1; for minimization problems, line 4 changes to “**if** $f(s') \leq f(s)$ **then**” accordingly.

Algorithm 9.1 (1+1)-GP Algorithm

Input: objective function $f : \mathcal{S} \rightarrow \mathbb{R}$ with the solution space \mathcal{S}

Output: solution from \mathcal{S}

Process:

- 1: let s = an initial solution from \mathcal{S} ;
 - 2: **while** criterion is not met **do**
 - 3: apply mutation on s to generate s' ;
 - 4: **if** $f(s') \geq f(s)$ **then**
 - 5: $s = s'$
 - 6: **end if**
 - 7: **end while**
 - 8: **return** s
-

SMO-GP is a simple multi-objective GP algorithm that simultaneously maximizes two or more objective functions [Neumann, 2012]. As presented in Algorithm 9.2, it repeatedly generates an offspring solution by mutation and maintains a set of non-dominated solutions created-so-far. If SMO-GP

is used for multi-objective minimization, notations “ \succ ” and “ \succeq ” in Algorithm 9.2 change to “ \prec ” and “ \preceq ”, respectively.

Algorithm 9.2 SMO-GP Algorithm

Input: objective vector $f = (f_1, f_2, \dots, f_m)$ with the solution space \mathcal{S}

Output: set of solutions from \mathcal{S}

Process:

- 1: let $s =$ an initial solution from \mathcal{S} ;
 - 2: let $P = \{s\}$;
 - 3: **while** criterion is not met **do**
 - 4: select a solution s from P uniformly at random;
 - 5: apply mutation on s to generate s' ;
 - 6: **if** $\nexists z \in P$ such that $z \succ s'$ **then**
 - 7: $P = (P \setminus \{z \in P \mid s' \succeq z\}) \cup \{s'\}$
 - 8: **end if**
 - 9: **end while**
 - 10: **return** P
-

The mutation operator in Definition 9.1 applies one of the three operators, i.e., *substitution*, *insertion* and *deletion*, uniformly at random and repeats this process for k times independently. For (1+1)-GP and SMO-GP using the mutation operator with $k = 1$, we denote them by (1+1)-GP-single and SMO-GP-single, respectively. The three operators are illustrated in Figure 9.1. Note that the arity of each function in F is 2 for simplicity.

Definition 9.1 (Mutation). *It applies one of the following three operators uniformly at random, and repeats this process for k times independently.*

[Substitution] Replace a randomly chosen leaf node of the solution with a new node selected uniformly at random from T .

[Insertion] Select a node v of the solution randomly, select a node u from F randomly, and select a node w from T randomly. Replace v with u whose children are v and w , the order of which is random.

[Deletion] Select a leaf node v of the solution randomly, whose parent and sibling are p and u , respectively. Replace p with u and delete p and v .

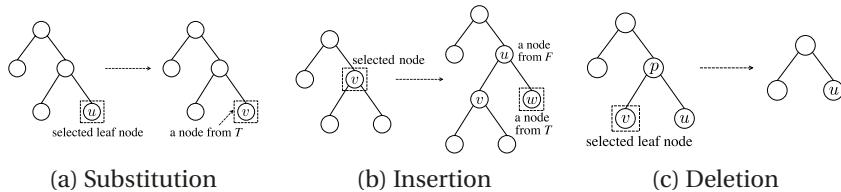


Figure 9.1. Illustration of the three operators in mutation.

(1+1)-GP and SMO-GP are similar to (1+1)-EA, i.e., Algorithm 2.1, and GSEMO, i.e., Algorithm 2.5 using bit-wise mutation, respectively. The main difference lies in the solution representation and the mutation operator.

The bloat problem, i.e., the solution complexity grows without quality increase, is often encountered in GP [Poli et al., 2008]. One way to tackle this problem is the parsimony approach, which prefers the solution with a smaller complexity when the compared solutions have the same fitness value; this is based on the Occam's razor philosophy popularly adopted in machine learning. For example, (1+1)-GP with the parsimony approach changes line 4 of Algorithm 9.1 to

“**if** ($f(s') > f(s)$) or ($f(s') = f(s) \wedge C(s') \leq C(s)$) **then**”,

where $C(s)$ is the complexity of the solution s . For the complexity measure $C(s)$ of a tree-structured solution, its number of nodes can be considered [Neumann, 2012, Wagner and Neumann, 2012].

In our analysis, we use the procedure in Definition 9.2 to construct an initial tree for (1+1)-GP and SMO-GP.

Definition 9.2 (Initial Tree Construction). *Given a set F of functions and a set T of terminals, the tree starts from a root node by selecting a node randomly from $F \cup T$, and then expands a node from F recursively by selecting its children randomly from $F \cup T$, until all leaf nodes are from T .*

9.2 Case Study: Maximum Matchings

In this section, we analyze the running time of (1+1)-GP until achieving an $(1/(1 + \epsilon))$ -approximation ratio for the maximum matching problem.

9.2.1 Solution Representation and Fitness Calculation

The maximum matching problem can be described as follows. Given an undirected graph $G = (V, E)$ on n vertices and m edges where V and E are the vertex set and edge set, respectively, a matching is a subset E' of the edge set E , such that there is no pair of edges in E' sharing a common vertex. The goal is to find a matching with the largest number of edges. For the convenience of discussion, let the edges be indexed as $\{1, 2, \dots, m\}$.

The maximum matching problem is one of the limited number of combinatorial optimization problems where the running time of EAs has been analyzed. In [Giel and Wegener, 2003, 2006], a solution is represented as a Boolean string of length m . For a solution $s \in \{0, 1\}^m$, $s_i = 1$ means that the edge i is selected by s , and $s_i = 0$ means that the edge i is not selected. The following fitness function has been used for maximization:

$$f(\mathbf{s}) = \sum_{i=1}^m s_i - c \cdot \sum_{v \in V} p(v, \mathbf{s}), \quad (9.1)$$

where $p(v, \mathbf{s}) = \max\{0, d(v, \mathbf{s}) - 1\}$, $d(v, \mathbf{s})$ is the degree of the vertex v on the subgraph represented by \mathbf{s} , and $c \geq m + 1$ is a penalty coefficient making any matching have a larger fitness value than any non-matching.

For GP on the maximum matching problem, we use $F = \{J\}$ and $L = \{1, 2, \dots, m\}$, where J is a joint function with arity 2 and L contains the m edges. The fitness of a tree-structured solution s can be calculated by the procedure in Definition 9.3.

Definition 9.3 (Fitness Calculation). *Given a tree-structured solution s , its fitness is calculated as follows:*

1. Parse the tree s in order and add the leaves to a list l ;
2. Parse the list l from left to right, and for each leaf add the corresponding edge to a set Q ;
3. Compute the fitness of the graph $G_Q = (V, Q)$.

In our analysis, the fitness function equivalent to Eq. (9.1) will be used. In other words, in step 3 of Definition 9.3, it calculates

$$f(s) = |Q| - c \cdot \sum_{v \in V} p(v, G_Q), \quad (9.2)$$

where $p(v, G_Q) = \max\{0, d(v, G_Q) - 1\}$, $d(v, G_Q)$ is the degree of the vertex v on the subgraph G_Q , and $c \geq m + 1$. For example, for the graph G in Figure 9.2(a), a maximum matching is an arbitrary edge; Figures 9.2(b-c) give the binary-vector representation and a tree-structured representation for a set of edges $\{e_1, e_2\}$, respectively. When computing the fitness of the tree-structured solution s , $l = \{1, 2, 1\}$, $Q = \{1, 2\}$, and then $f(s) = 2 - c \cdot (0+0+1) = 2 - c$. Note that the edges 1, 2, 3 correspond to the edges e_1, e_2, e_3 on the graph G , respectively, and the weights w_1, w_2, w_3 are not used here, but will be used in the following analysis on minimum spanning trees.

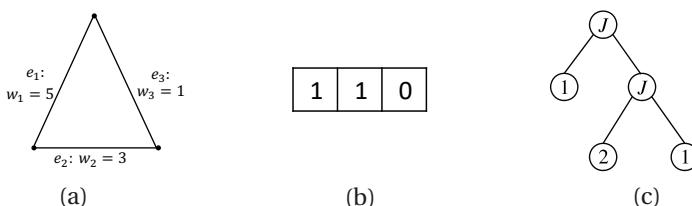


Figure 9.2. An example of (a) a graph G , (b) a binary-vector solution containing the edges $\{e_1, e_2\}$ and (c) a tree-structured solution s containing the edges $\{e_1, e_2\}$.

9.2.2 Analysis of (1+1)-GP

We now analyze the expected running time of (1+1)-GP with the parsimony approach on the maximum matching problem. Lemma 9.1 presents the properties of the number of leaves in the initial tree. The proof is provided in Appendix A.4.

Lemma 9.1. *For the initial tree construction in Definition 9.2, let T_{init} denote the number of leaves in the initial tree. Then, $\mathbb{E}[T_{init}] = m/(m - 1)$ and $\mathbb{E}[T_{init}^2] \leq m/(m - 3)$.*

For a subset Q of the whole edge set E , a vertex is *free* if it is not the endpoint of any edge in Q ; an edge (v_i, v_j) is free if it does not belong to Q and v_i, v_j are free vertices; an edge is *matching* if it belongs to Q and $d(v_i, G_Q) = d(v_j, G_Q) = 1$. For a matching M , an augmenting path with respect to M is a path v_1, v_2, \dots, v_k of odd length, i.e., with even k , where the edges $(v_{2i}, v_{2i+1}) \forall i \in [k/2 - 1]$ belong to M whereas the other edges do not, and v_1, v_k are free vertices. Lemma 9.2 gives an upper bound of the length of an augmenting path.

Lemma 9.2. *[Neumann and Witt, 2010] Let $G = (V, E)$ be a graph. Let M and M^* denote a non-maximum matching and a maximum matching, respectively. There exists an augmenting path with respect to M whose length is bounded from above by $2\lfloor |M|/(|M^*| - |M|) \rfloor + 1$.*

Theorem 9.1 gives a running time upper bound of (1+1)-GP-single with the parsimony approach for achieving an $(1/(1 + \epsilon))$ -approximation ratio. Before giving the proof, we briefly introduce the main proof idea. We first analyze the running time until finding a matching by using the multiplicative drift analysis approach in Theorem 2.4; then, we derive the running time for a non-maximum matching improving to an $(1/(1 + \epsilon))$ -optimal matching. In the improving process, we use the property that swapping the edges of an augmenting path of a non-maximum matching can increase the number of matching edges, which has been used for analyzing (1+1)-EA on maximum matchings [Giel and Wegener, 2003].

Theorem 9.1. *For (1+1)-GP-single with the parsimony approach solving the maximum matching problem, the expected running time until finding an $(1/(1 + \epsilon))$ -optimal matching is $O((3m \min\{m, n\}/2)^{\lceil \epsilon^{-1} \rceil})$, where $\epsilon > 0$.*

Proof. We divide the optimization process into two phases:

- *phase 1*: starts after initialization and finishes until finding a solution which represents a matching M and has exactly $|M|$ leaves.
- *phase 2*: starts after phase 1 and finishes until finding a solution which represents an $(1/(1 + \epsilon))$ -optimal matching.

We first analyze each phase i and derive an upper bound E_i of the expected

running time. By summing them up, we get an upper bound ($E_1 + E_2$) of the expected running time of the whole optimization process.

In phase 1, let s_t ($t \geq 0$) denote the solution after t generations of (1+1)-GP-single. We use multiplicative drift analysis in Theorem 2.4 to derive a running time upper bound of this phase. Note that ξ_t in Theorem 2.4 is just s_t . We design the distance function as $\forall s : V(s) = \sum_v p(v, s) + \sum_{e \in s} (N(e) - 1)$, where $p(v, s) = p(v, G_Q)$ in Eq. (9.2), $G_Q = (V, Q)$ is the graph generated in Definition 9.3 when calculating the fitness of s , $e \in s$ means $e \in Q$, and $N(e)$ denotes the number of occurrences of edge e in the leaves of the solution s . It can be verified that $V(s) = 0$ iff a matching is found and there is no duplicate edge in the solution, i.e., the goal of this phase is reached. We are then to analyze $\mathbb{E}[V(s_t) - V(s_{t+1}) | \xi_t = s_t]$.

We first show that $V(s_t)$ is non-increasing. The term $\sum_v p(v, s_t)$ never increases because this term dominates the fitness function f in Eq. (9.2) and f never decreases. Consider three possible operators in one mutation step. (1) by deletion, $\forall e \in s_t$, $N(e)$ cannot increase, leading to the fact that $V(s_t)$ cannot increase.

(2) by insertion, only free edges can be accepted. This will add a new edge e with $N(e) = 1$, and the term $\sum_{e \in s_t} (N(e) - 1)$ does not increase. Thus, $V(s_t)$ cannot increase.

(3) by substitution, it can be viewed as deletion first and then insertion. If an edge e with $N(e) > 1$ is deleted, $\sum_{e \in s_t} (N(e) - 1)$ decreases by 1, and any following insertion cannot increase $V(s_t)$, because it can increase $\sum_{e \in s_t} (N(e) - 1)$ by at most 1. The only possible way to increase $\sum_{e \in s_t} (N(e) - 1)$ is to delete an edge e with $N(e) = 1$ and insert an edge e' with $N(e') \geq 1$, which will increase $\sum_{e \in s_t} (N(e) - 1)$ by 1. We consider the influence on $\sum_v p(v, s_t)$ by this event. Inserting an edge e' with $N(e') \geq 1$ will not affect $\sum_v p(v, s_t)$. For deleting an edge e with $N(e) = 1$, if it is a matching edge, $\sum_v p(v, s_t)$ will not be affected whereas $|Q|$ decreases by 1, and thus, the fitness in Eq. (9.2) decreases and the offspring solution will be rejected; otherwise, the deleted edge e shares endpoints with other edges, and its deletion will decrease $\sum_v p(v, s_t)$ by at least 1, leading to the fact that $V(s_t)$ will not increase.

Next, we show that in every mutation step, there exist at least $\lceil V(s_t)/4 \rceil$ leaves whose deletion will be accepted and decrease $V(s_t)$ by at least 1, i.e., $V(s_t) - V(s_{t+1}) \geq 1$. If $\sum_{e \in s_t} (N(e) - 1) > \lfloor V(s_t)/2 \rfloor$, deleting one of these duplicate edges will not affect the fitness while decreasing the complexity of the solution, and thus, $V(s_t)$ decreases by 1. If $\sum_{e \in s_t} (N(e) - 1) \leq \lfloor V(s_t)/2 \rfloor$, i.e., $\sum_v p(v, s_t) \geq \lceil V(s_t)/2 \rceil$, there are at least $\lceil \sum_v p(v, s_t)/2 \rceil \geq \lceil V(s_t)/4 \rceil$ edges contributing to the value $\sum_v p(v, s_t)$. For one of these edges e , if $N(e) = 1$, deleting it will decrease $\sum_v p(v, s_t)$ by at least 1; if $N(e) > 1$, deleting it will not affect $\sum_v p(v, s_t)$ while decreasing $\sum_{e \in s_t} (N(e) - 1)$ by 1. Thus, our claim holds. Let L_t denote the number of leaves of the solution s_t . The probability of decreasing $V(s_t)$ by one step is at least $(1/3) \cdot (V(s_t)/4)/L_t$, where $1/3$ is the probability of applying deletion in mutation.

Based on the above analysis of $V(s_t)$, we can derive

$$\mathbb{E}[V(s_t) - V(s_{t+1}) \mid \xi_t = s_t] \geq \frac{V(s_t)}{12L_t}. \quad (9.3)$$

We then give an upper bound of L_t . Note that $\sum_{e \in s_t} (N(e) - 1)$ is the number of duplicate edges contained in the leaves of s_t , and $\sum_v p(v, s_t)$ gives the number of edges whose deletion on G_Q can generate a matching. Because the number of edges of a matching is no larger than $\min\{m, n\}$, the number of leaves is upper bounded by $V(s_t) + \min\{m, n\}$; that is, $L_t \leq V(s_t) + \min\{m, n\}$. Because $V(s_0) \leq \sum_{e \in s_0} (N(e) - 1) + 2(T_{init} - \sum_{e \in s_0} (N(e) - 1)) \leq 2T_{init}$ and $V(s_t)$ is non-increasing, we have $V(s_t) \leq 2T_{init}$, leading to $L_t \leq 2T_{init} + \min\{m, n\}$. Thus,

$$\mathbb{E}[V(s_t) - V(s_{t+1}) \mid \xi_t = s_t] \geq \frac{V(s_t)}{12(2T_{init} + \min\{m, n\})}. \quad (9.4)$$

By Theorem 2.4 and $V_{min} = \min\{V(s_t) \mid V(s_t) > 0\} = 1$, we have

$$\mathbb{E}[\tau \mid \xi_0 = s_0] \leq 12(2T_{init} + \min\{m, n\})(1 + \log(V(s_0))). \quad (9.5)$$

Due to $V(s_0) \leq 2T_{init}$, we have

$$E_1 = O((T_{init} + \min\{m, n\}) \log T_{init}). \quad (9.6)$$

In phase 2, the solution will always be a matching, because a non-matching has a smaller fitness than a matching, and also, the solution has no duplicate edges because inserting a duplicate edge will increase the complexity while not affecting the fitness, and substituting an existing edge with a duplicate edge will decrease the fitness by 1. The number of leaves of the solution in this phase is always no larger than $\min\{m, n\}$, implying that the probability of a specific substitution in a mutation step is at least $(1/3) \cdot (1/(m \cdot \min\{m, n\}))$, where $1/3$ is the probability of applying substitution in mutation, $1/\min\{m, n\}$ is a lower bound of the probability of deleting a specific leaf of the solution, and $1/m$ is the probability of selecting a specific edge from T for insertion.

Let M be the matching represented by the current solution and M^* be a maximum matching. By Lemma 9.2, there exists an augmenting path l with length bounded above by $2\lfloor |M| / (|M^*| - |M|) \rfloor + 1$. As the goal is to find an $(1/(1 + \epsilon))$ -optimal matching, we have $(1 + \epsilon)|M| < |M^*|$ before this phase finishes. Thus, $|l| \leq 2\lceil \epsilon^{-1} \rceil - 1$, where $|l|$ denotes the length of l . Note that $|l|$ is odd. To improve the current matching, i.e., to increase $|M|$, we can delete all the edges on l which belong to M and insert all the edges on l which do not belong to M . Such behavior will increase $|M|$ by 1, by the definition of augmenting path. To achieve this, we can exchange the first or last two edges of l by substitution in one step, then repeat this process for $\lfloor |l|/2 \rfloor$ times, and finally insert the remaining edge on l . Note

that each of the $\lfloor |l|/2 \rfloor$ substitution is accepted because it does not affect the fitness and the complexity; the last insertion is also accepted as it increases the fitness by 1. We call such a procedure a *successful phase*, the probability of which is at least $(2 \cdot (1/(3m \cdot \min\{m, n\})))^{\lfloor |l|/2 \rfloor} \cdot (1/(3m)) = \Omega(2^{\lfloor |l|/2 \rfloor}/((3m)^{\lceil |l|/2 \rceil}(\min\{m, n\})^{\lfloor |l|/2 \rfloor}))$, where the factor 2 holds because there are two choices for substitution; that is, exchanging the first two edges of an augmenting path or the last two. Thus, the expected number of phases until a successful phase is $O((3m)^{\lceil |l|/2 \rceil}(\min\{m, n\}/2)^{\lfloor |l|/2 \rfloor})$, leading to an upper bound $O(\lceil |l|/2 \rceil \cdot (3m)^{\lceil |l|/2 \rceil}(\min\{m, n\}/2)^{\lfloor |l|/2 \rfloor})$ of the expected number of steps until success. We then show a tighter upper bound by making full use of the fact that these $O((3m)^{\lceil |l|/2 \rceil}(\min\{m, n\}/2)^{\lfloor |l|/2 \rfloor})$ phases are unsuccessful. It can be seen that a phase can continue successfully with probability at most $1/(3m)$. Thus, the expected number of steps for an unsuccessful phase is $O(1)$, implying that the expected number of steps until success is $O((3m)^{\lceil |l|/2 \rceil}(\min\{m, n\}/2)^{\lfloor |l|/2 \rfloor} + \lceil |l|/2 \rceil) = O((3m)^{\lceil |l|/2 \rceil}(\min\{m, n\}/2)^{\lfloor |l|/2 \rfloor})$, where the term $\lceil |l|/2 \rceil$ after “+” is the length of the final successful phase. The rigorous proof for this tighter upper bound can be found in Appendix C of [Giel and Wegener, 2002]. As $|M^*| \leq \min\{m, n\}$, it requires at most $\min\{m, n\}$ successes to find an $(1/(1 + \epsilon))$ -optimal matching. Thus, the expected running time for this phase is at most

$$\begin{aligned} E_2 &= O\left((3m)^{\lceil |l|/2 \rceil} \left(\frac{\min\{m, n\}}{2}\right)^{\lceil |l|/2 \rceil}\right) \\ &= O\left(\left(\frac{3m \min\{m, n\}}{2}\right)^{\lceil \epsilon^{-1} \rceil}\right). \end{aligned} \quad (9.7)$$

The expected running time of the whole process is at most

$$\begin{aligned} E_1 + E_2 \\ = O\left((T_{init} + \min\{m, n\}) \log T_{init} + \left(\frac{3m \min\{m, n\}}{2}\right)^{\lceil \epsilon^{-1} \rceil}\right), \end{aligned} \quad (9.8)$$

which is a random variable depending on T_{init} . Because the initial tree is constructed by Definition 9.2, we have $\mathbb{E}[T_{init}] = m/(m - 1)$ by Lemma 9.1; $\mathbb{E}[\log T_{init}] \leq \log \mathbb{E}[T_{init}] = \log(m/(m - 1))$ by Jensen's inequality and the concavity of the function $\log x$; $\mathbb{E}[T_{init} \log T_{init}] \leq \mathbb{E}[T_{init}^2] \leq m/(m - 3)$ by Lemma 9.1. Thus, the expected running time over all possible initial trees is $O((3m \min\{m, n\}/2)^{\lceil \epsilon^{-1} \rceil})$. \square

Therefore, (1+1)-GP requires $O((3m \min\{m, n\}/2)^{\lceil \epsilon^{-1} \rceil})$ expected running time to achieve an $(1/(1 + \epsilon))$ -approximation ratio for the maximum matching problem. Compared with the expected running time $O(m^{2\lceil \epsilon^{-1} \rceil})$ of (1+1)-EA [Giel and Wegener, 2003], (1+1)-GP has a better upper bound.

By comparing our analysis with the analysis of (1+1)-EA in [Giel and Wegener, 2003], we find that the efficiency of (1+1)-GP is due to the larger probability of exchanging two edges on an augmenting path in the process of improving a non-maximum matching. For (1+1)-EA, exchanging two edges is by flipping the bits on the corresponding two positions, occurring with probability $\Theta(1/m^2)$; for (1+1)-GP, exchanging two edges is by substitution, which deletes a specific edge from the current solution with probability at least $1/\min\{m, n\}$ and then inserts a specific edge from T with probability $1/m$. The large probability of deletion for (1+1)-GP owes to the fact that in phase 2 of the optimization process, the evolved solution always represents a matching and has no duplicate edges, making its number of leaves no larger than $\min\{m, n\}$. Thus, our analysis suggests that the variable solution structure can be helpful for evolutionary optimization if the solution complexity is well controlled.

9.3 Case Study: Minimum Spanning Trees

In this section, we analyze the running time of (1+1)-GP and SMO-GP for solving the minimum spanning tree problem, respectively.

9.3.1 Solution Representation and Fitness Calculation

The minimum spanning tree problem can be described as follows. Given an undirected connected graph $G = (V, E)$ on n vertices and m edges where V and E are the vertex set and edge set respectively, the goal is to find a connected subgraph $G' = (V, E' \subseteq E)$ with the minimum sum of edge weights. For the convenience of discussion, let the edges be indexed as $\{1, 2, \dots, m\}$ with non-negative weights $\{w_1, w_2, \dots, w_m\}$. Let w_{\max} denote the maximum weight, i.e., $w_{\max} = \max\{w_i \mid 1 \leq i \leq m\}$.

For EAs solving the minimum spanning tree problem, Neumann and Wegener [2007] represent a solution by a Boolean string s of length m , i.e., $s \in \{0, 1\}^m$, where $s_i = 1$ means that the edge i is selected by s . The following fitness function has been used for minimization:

$$f(s) = (c(s) - 1)w_{ub}^2 + \left(\sum_{i=1}^m s_i - n + 1 \right) w_{ub} + \sum_{i=1}^m w_i s_i, \quad (9.9)$$

where $c(s)$ is the number of connected components of the subgraph described by s , and $w_{ub} = n^2 w_{\max}$. Note that in this fitness function, the first term $(c(s) - 1)w_{ub}^2$ enforces a subgraph with fewer connected components to be better, the second term $(\sum_{i=1}^m s_i - n + 1)w_{ub}$ enforces a connected subgraph with fewer edges to be better, and the last term $\sum_{i=1}^m w_i s_i$ enforces a spanning tree with a smaller weight to be better.

For GP on the minimum spanning tree problem, we use $F = \{J\}$ and $L = \{1, 2, \dots, m\}$, where J is a joint function with arity 2 and L contains the m edges. The fitness of a tree-structured solution s can be calculated by the procedure in Definition 9.3. The fitness function equivalent to Eq. (9.9) will be used; that is, in step 3 of Definition 9.3, it calculates

$$W(s) = (c(G_Q) - 1)w_{ub}^2 + (|Q| - (n - 1))w_{ub} + \sum_{i \in Q} w_i, \quad (9.10)$$

where $c(G_Q)$ denotes the number of connected components in the graph G_Q . For example, when computing the fitness of the solution s for the graph G in Figure 9.2, $l = \{1, 2, 1\}$, $Q = \{1, 2\}$, and $W(s) = (1 - 1) \cdot w_{ub}^2 + (2 - 2) \cdot w_{ub} + w_1 + w_2 = 8$.

9.3.2 Analysis of (1+1)-GP

First, we analyze the running time of (1+1)-GP-single with the parsimony approach on the minimum spanning tree problem. Assume that all edge weights are integers, as in the previous analyses of EAs on the minimum spanning tree problem [Neumann and Wegener, 2006, 2007, Doerr et al., 2012c]. We will not distinguish a solution and the subgraph G_Q that it represents for convenience.

Lemma 9.3 gives a property on local changes of spanning trees.

Lemma 9.3. [Neumann and Wegener, 2007] *Let M describe a non-minimum spanning tree. There exist $k \in [n - 1]$ different two-edge exchanges which delete one edge in M and insert one edge not in M , such that each two-edge exchange generates a spanning tree with a smaller weight than M , and the average weight decrease of these k exchanges is $(W_M - W_{opt})/k$, where W_M and W_{opt} are the weights of M and a minimum spanning tree, respectively.*

Theorem 9.2 shows that (1+1)-GP-single can solve the minimum spanning tree problem in $O(mn(\log n + \log w_{\max}))$ expected running time.

Theorem 9.2. *For (1+1)-GP-single with the parsimony approach solving the minimum spanning tree problem, the expected running time is $O(mn(\log n + \log w_{\max}))$.*

Proof. Inspired from the analysis of (1+1)-EA on the minimum spanning tree problem [Neumann and Wegener, 2007], we divide the optimization process into three phases:

- *phase 1*: starts after initialization and finishes until finding a solution representing a connected subgraph.
- *phase 2*: starts after phase 1 and finishes until finding a solution with $(n - 1)$ leaves which represents a spanning tree.
- *phase 3*: starts after phase 2 and finishes until finding a solution representing a minimum spanning tree.

We first analyze each phase i and derive an upper bound E_i of the expected running time. By summing them up, we get an upper bound $(E_1 + E_2 + E_3)$ of the expected running time of the whole process.

In phase 1, let c denote the number of connected components of the current solution. c cannot increase, because a solution with more connected components has a larger fitness value by the definition of $W(s)$ in Eq. (9.10). For a subgraph with c connected components, there must exist at least $(c - 1)$ edges whose insertion decreases the number of connected components by 1, because the original graph is connected. Thus, the probability of decreasing c by 1 in one step of (1+1)-GP-single is at least $(1/3) \cdot (c - 1)/m$, where $1/3$ is the probability of applying insertion in mutation, and $(c - 1)/m$ is the probability of selecting one of those $(c - 1)$ edges from T for insertion. This implies that the expected number of steps for decreasing c by 1 is at most $3m/(c - 1)$. By $c \leq n$, the expected running time until finding a connected subgraph is at most

$$E_1 = \sum_{c=2}^n \frac{3m}{c - 1} = O(m \log n). \quad (9.11)$$

In phase 2, let l denote the number of leaves of the current solution. Note that in this phase, the solution is always connected, because a non-connected subgraph has a larger fitness than a connected one by Eq. (9.10). l cannot increase, because the insertion of a new leaf u will increase the fitness $W(s)$ if u represents a new edge, or will keep $W(s)$ unchanged but increase the complexity $C(s)$ if u represents an edge in the solution. For a solution with l leaves which represents a connected subgraph, there exist at least $(l - (n - 1))$ leaves whose deletion decreases l by 1 and keeps the subgraph connected, because it contains at least one spanning tree and a spanning tree contains exactly $(n - 1)$ number of edges. Thus, the probability of decreasing l by 1 in one step is at least $(1/3) \cdot (l - n + 1)/l$, where $1/3$ is the probability of applying deletion in mutation, and $(l - n + 1)/l$ is the probability of selecting one of those $(l - n + 1)$ leaves from all l leaves for deletion. Let l_{init} denote the number of leaves of the solution when this phase starts. The expected running time until l decreases to $(n - 1)$ is at most $\sum_{l=n}^{l_{init}} 3l/(l - n + 1)$. Note that $l = n - 1$ implies that the current solution is a spanning tree, as the solution is always connected. Thus, we have

$$E_2 = \sum_{l=n}^{l_{init}} \frac{3l}{l - n + 1}. \quad (9.12)$$

We then derive an upper bound of l_{init} . The number of leaves can increase by insertion only. In the first phase of finding a connected subgraph, insertion can be accepted only if it can decrease the number of connected components by 1; otherwise, it will be rejected because it will either increase the number of edges or keep the edges but increase the complexity.

Because the number of connected components can decrease for at most $(n - 1)$ times, insertion can be accepted by at most $(n - 1)$ times in phase 1. Thus, $l_{init} \leq T_{init} + n - 1$, implying

$$E_2 \leq \sum_{l=n}^{T_{init}+n-1} \frac{3l}{l-n+1} = O(T_{init} + n \log T_{init}). \quad (9.13)$$

In phase 3, the solution will always be a spanning tree, because a non-spanning tree has a larger fitness than a spanning tree. After phase 2, the spanning tree found has exactly $(n - 1)$ leaves, leading to the fact that insertion and deletion in phase 3 will not be accepted, because insertion will lead to more edges or the same edges with a larger complexity, whereas deletion will lead to more connected components. Thus, the number of leaves of the solution during this phase is always $n - 1$.

Next we prove the expected running time until finding a minimum spanning tree by using multiplicative drift analysis in Theorem 2.4. Let s_t denote the solution after t generations in this phase. ξ_t in Theorem 2.4 is just s_t . We construct the distance function as $\forall s : V(s) = W(s) - W_{opt}$. It can be verified that $V(s) = 0$ iff a minimum spanning tree is found, i.e., the goal of this phase is reached. For the expected drift, we have $\mathbb{E}[V(s_t) - V(s_{t+1}) | \xi_t = s_t] = W(s_t) - \mathbb{E}[W(s_{t+1}) | s_t]$. $W(s_t)$ is non-increasing with t , because a spanning tree with a larger weight will be rejected. According to Lemma 9.3, for a solution s representing a spanning tree, there exist k accepted substitutions whose average weight decrease is $(W(s) - W_{opt})/k$ for some $k \in [n - 1]$. Note that the probability of a specific substitution is $(1/3) \cdot (1/(m(n-1)))$, where $1/3$ is the probability of applying substitution in mutation, $1/(n-1)$ is the probability of deleting a specific edge from the $(n - 1)$ leaves, and $1/m$ is the probability of selecting one of the m edges from T for insertion. Thus, we have

$$\mathbb{E}[V(s_t) - V(s_{t+1}) | \xi_t = s_t] \geq \frac{k}{3m(n-1)} \cdot \frac{W(s_t) - W_{opt}}{k} = \frac{1}{3m(n-1)} V(s_t). \quad (9.14)$$

By Theorem 2.4 and $\min\{V(s_t) \mid V(s_t) > 0\} \geq 1$, we have $\mathbb{E}[\tau | \xi_0 = s_0] \leq 3m(n-1)(1+\log(V(s_0)))$. Considering that this phase starts from a spanning tree, we have $V(s_0) \leq nw_{max}$, and thus,

$$E_3 = O(mn(\log n + \log w_{max})). \quad (9.15)$$

Using $\mathbb{E}[T_{init}] = m/(m - 1)$ in Lemma 9.1, the expected running time of the whole process is at most

$$\mathbb{E}[E_1 + E_2 + E_3] = O(mn(\log n + \log w_{max})). \quad (9.16)$$

□

Thus, we have proved that (1+1)-GP solves the minimum spanning tree problem in $O(mn(\log n + \log w_{\max}))$ expected running time. Compared with the expected running time $O(m^2(\log n + \log w_{\max}))$ of (1+1)-EA [Neumann and Wegener, 2007, Doerr et al., 2012c], (1+1)-GP has a better upper bound. In [Neumann and Wegener, 2007], it has also been proved that (1+1)-EA needs $\Theta(n^4 \log n)$ expected running time for an example graph with $m = \Theta(n^2)$ and $w_{\max} = n^2$. From Theorem 9.2, we can conclude that (1+1)-GP is strictly better than (1+1)-EA on this example, because the expected running time of (1+1)-GP is upper bounded by $O(n^3 \log n)$.

By comparing our analysis with the analysis of (1+1)-EA [Neumann and Wegener, 2007], we find a similar reason why (1+1)-GP is more efficient as what has been found from our analysis on the maximum matching problem. This is because in the process of improving a non-minimum spanning tree, i.e., phase 3, (1+1)-EA needs to flip two specific bits for a good substitution, occurring with probability $\Theta(1/m^2)$; (1+1)-GP can perform such a substitution by deleting a specific edge from the current solution with probability $1/(n - 1)$ and then inserting a specific edge from T with probability $1/m$. The large probability of deletion for (1+1)-GP owes to the fact that the solution in phase 3 always has exactly $(n - 1)$ leaves. Thus, the analysis on the minimum spanning tree problem suggests that the variable solution structure can be helpful if the solution complexity is properly controlled.

9.3.3 Analysis of SMO-GP

The minimum spanning tree problem can be solved in $O(mn(n + \log w_{\max}))$ expected running time by the multi-objective reformulation [Neumann and Wegener, 2006], which transforms the original problem to a bi-objective optimization problem by introducing an auxiliary objective and employs GSEMO to solve the transformed problem. In this section, we analyze SMO-GP solving the multi-objective reformulated minimum spanning tree problem MO-MST by treating the complexity of a solution as the auxiliary objective. Note that our analysis does not require the edge weights to be integers.

Definition 9.4 (MO-MST). *The bi-objective minimum spanning tree problem MO-MST is defined as*

$$\text{MO-MST}(s) = (W(s), C(s)), \quad (9.17)$$

where $C(s)$ is the complexity of a solution s , i.e., the number of nodes in the GP-tree of s .

For MO-MST, in step 3 of Definition 9.3, besides computing $W(s)$, it also needs to compute the complexity $C(s) = 2 \cdot |\mathcal{U}| - 1$. For example, the solution s in Figure 9.2 has the complexity $C(s) = 2 \cdot 3 - 1 = 5$.

For a solution s with i leaves where $0 \leq i \leq n - 1$, the subgraph represented by s contains at most i edges, and thus contains at least $(n - i)$

connected components. Because the number of connected components dominates the fitness function $W(s)$ in Eq. (9.10), this solution will have the minimum $W(s)$ value, denoted as W_i , when the subgraph that it represents has $(n - i)$ connected components and its weight is the minimum among all subgraphs with $(n - i)$ connected components. In other words, W_i equals $(n - i - 1)w_{ub}^2 + (i - (n - 1))w_{ub}$ plus the minimum weight among all subgraphs with $(n - i)$ connected components. Specifically, for a solution with $(n - 1)$ leaves, the minimum $W(s)$ value, i.e., W_{n-1} , is just the weight of a minimum spanning tree, which is also the minimum $W(s)$ value for all solutions. This implies that such a solution will dominate any solution with more than $(n - 1)$ leaves, because its complexity $C(s)$ is smaller and its $W(s)$ value is not larger. Note that W_i decreases as i increases.

Based on the above analysis, the Pareto front of MO-MST is $\{(W_0, 0)\} \cup \{(W_i, 2i - 1) \mid i \in [n - 1]\}$. For $(W_0, 0)$, the corresponding solution is the empty tree. For $(W_i, 2i - 1)$, the corresponding solution is a tree with i leaves which constitutes a subgraph with the minimum weight among all subgraphs with $(n - i)$ connected components. The Pareto optimal solution with objective value $(W_{n-1}, 2n - 3)$ represents a minimum spanning tree.

Lemma 9.4 gives an upper bound of the population size of SMO-GP-single. The proof is provided in Appendix A.4.

Lemma 9.4. *For SMO-GP-single solving the MO-MST problem, the population size is upper bounded by $T_{init} + n$.*

Theorem 9.3 shows that SMO-GP-single solves the minimum spanning tree problem in $O(mn^2)$ expected running time. Compared with the expected running time $O(mn(n + \log w_{\max}))$ of GSEMO [Neumann and Weicker, 2006], SMO-GP-single has a better upper bound.

Theorem 9.3. *For SMO-GP-single solving the MO-MST problem, the expected running time is $O(mn^2)$.*

Proof. Inspired from the analysis of GSEMO solving the multi-objective formulation of the minimum spanning tree problem [Neumann and Weicker, 2006], we divide the optimization process into two phases:

- *phase 1*: starts after initialization and finishes until finding the empty graph, i.e., the objective vector $(W_0, 0)$.

- *phase 2*: starts after phase 1 and finishes until finding the Pareto front.

We first derive a running time upper bound E_i for each phase i , and then sum them up to get an upper bound $(E_1 + E_2)$ of the expected running time of the whole evolutionary process.

In phase 1, let l denote the minimum number of leaves of solutions in the current population, and let s^* denote the corresponding solution with l leaves. l will never increase because a solution cannot be weakly dominated by another one with more leaves. By applying deletion on s^* , the offspring solution will always be accepted, because its second objective value, i.e., the

complexity, becomes the smallest and then it is not dominated by any solution in the current population; thus l decreases by 1. After applying such step for l times, the empty graph will be found. The probability of applying deletion on s^* is at least $(1/(T_{init} + n)) \cdot (1/3)$, because the probability of selecting a specific solution for mutation is at least $1/(T_{init} + n)$ by the population size no larger than $(T_{init} + n)$ in Lemma 9.4, and the probability of applying deletion is $1/3$. Thus, we have

$$E_1 = l \cdot 3(T_{init} + n) = T_{init} \cdot 3(T_{init} + n), \quad (9.18)$$

where the last equality holds by $l = T_{init}$ for the initial population.

In phase 2, let s_i^* denote the Pareto optimal solution with the first objective value W_i . From a best subgraph, i.e., a subgraph having the minimum weight, with k connected components, a best subgraph with $(k - 1)$ connected components can be generated by inserting the lightest edge which will not lead to a cycle. Using this property, s_{i+1}^* can be generated by inserting the lightest edge which will not lead to a cycle into s_i^* . Because the probability of selecting a specific solution for mutation is at least $1/(T_{init} + n)$ by Lemma 9.4, the probability of applying insertion is $1/3$, and the probability of selecting a specific edge from T for insertion is $1/m$, s_{i+1}^* can be generated in one step with probability at least $1/(3m(T_{init} + n))$ after s_i^* has been found. Note that s_0^* has been found when this phase starts and a Pareto optimal solution will be never lost once it is found. Thus, we have

$$E_2 = (n - 1) \cdot 3m(T_{init} + n). \quad (9.19)$$

By $\mathbb{E}[T_{init}] = m/(m - 1)$ and $\mathbb{E}[T_{init}^2] \leq m/(m - 3)$ in Lemma 9.1, the expected running time of the whole process is at most

$$\mathbb{E}[E_1 + E_2] = O(mn^2). \quad (9.20)$$

□

By comparing our analysis with the analysis of GSEMO [Neumann and Wegener, 2006], we find that the better performance of SMO-GP owes to the fact that the complexity objective leads to a more efficient process for finding the empty graph in phase 1 of the optimization process. Thus, the variable solution structure of GP allows us to have a complexity measure for the solution, and the optimization process is accelerated by minimizing the complexity directly as an auxiliary objective. This also suggests that if the solution complexity can be controlled, the variable solution structure can be helpful for optimization.

9.4 Empirical Verification

When comparing GP with GA on the maximum matching and minimum spanning tree problems, the running time bounds may not be that tight,

making the comparison not quite strict. Therefore, we conduct experiments to verify the theoretical analyses.

For the maximum matching problem, we compare (1+1)-GP with (1+1)-EA on the graphs with the number of edges $m = \Theta(n)$, $\Theta(n\sqrt{n})$, and $\Theta(n^2)$, respectively, briefly called as *sparse*, *moderate* and *dense* graphs, respectively. Let v_1, v_2, \dots, v_n denote the n nodes.

sparse graph: we use the cyclic graph where v_1 is connected with v_n and v_2 , v_i with $1 < i < n$ is connected with v_{i-1} and v_{i+1} , and v_n is connected with v_{n-1} and v_1 . Thus, $m = n$.

moderate graph: we use the graph where v_i with $1 \leq i \leq n - \lfloor \sqrt{n} \rfloor$ is connected with $v_{i+1}, v_{i+2}, \dots, v_{i+\lfloor \sqrt{n} \rfloor}$. Thus, $m = (n - \lfloor \sqrt{n} \rfloor) \lfloor \sqrt{n} \rfloor$.

dense graph: we use the complete graph where each node is connected with all other nodes. Thus, $m = n(n - 1)/2$.

For each kind of graph, the cardinality of a maximum matching is $\lfloor n/2 \rfloor$. The range of n is set to $[5, 50]$. On each n , we repeat the independent runs of each algorithm for 1,000 times, and the average running time is recorded as an estimation of the expected running time; we also record the standard deviation of the running time. For the approximation, we set ϵ to 0.1, 0.5 and 1, respectively.

The results with different ϵ values are plotted in Figures 9.3-9.5, respectively. It can be observed that the curve of (1+1)-GP is much lower than that of (1+1)-EA in Figures 9.3-9.5(b-c); their curves are a little close in Figures 9.3-9.5(a). By comparing the running time bounds in the second column of Table 9.1, we derive that (1+1)-GP performs as similarly as (1+1)-EA for the sparse graph with $m = \Theta(n)$, whereas (1+1)-GP is better for the moderate graph with $m = \Theta(n\sqrt{n})$ and the dense graph with $m = \Theta(n^2)$. Thus, the empirical results are consistent with theoretical analysis.

From Figures 9.5(a-b), it can be observed that there are two peaks at $n = 21, 43$. Note that the size of a maximum matching is $\lfloor n/2 \rfloor$. The $(1/(1 + \epsilon))$ -optimal matching to find has to contain at least $NE = \lceil \lfloor n/2 \rfloor / (1 + \epsilon) \rceil$ number of edges. For $\epsilon = 0.1$, when $n \in [0, 21]$, $NE = \lfloor n/2 \rfloor$; when $n \in [22, 43]$, $NE = \lfloor n/2 \rfloor - 1$; when $n \in [44, 65]$, $NE = \lfloor n/2 \rfloor - 2$. In summary, when $n \in [22k, 22(k + 1) - 1] \forall k \geq 0$, $NE = \lfloor n/2 \rfloor - k$. By the relation between NE and $\lfloor n/2 \rfloor$, it can be expected that the running time increases with n when $n \in [22k, 22(k + 1) - 1]$, because it is always to find a matching with k edges less than a maximum matching and the problem becomes harder as the number of nodes increases; the running time has a sudden decrease at $n = 22(k + 1)$, because it is currently to find a matching with $(k + 1)$ edges less than a maximum matching and the problem becomes easier than that for $n = 22(k + 1) - 1$. Thus, the empirical observation from Figures 9.5(a-b) is consistent with our analysis. For other figures, we do not observe the obvious sudden decrease in the running time; this may owe to the fact that the decrease of problem hardness caused by the increase of $(\lfloor n/2 \rfloor - NE)$ is not that apparent as the increase of problem hardness caused by the increase in the number n of nodes.

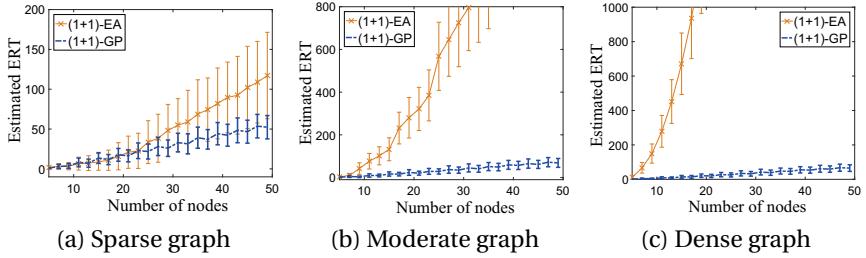


Figure 9.3. Estimated expected running time (ERT) comparison on the maximum matching problem for different graphs with $\epsilon = 1$.

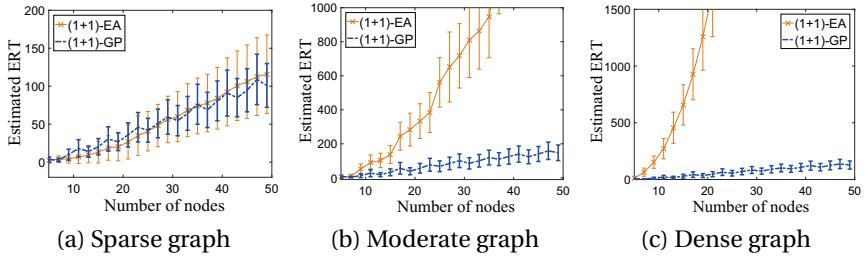


Figure 9.4. Estimated expected running time (ERT) comparison on the maximum matching problem for different graphs with $\epsilon = 0.5$.

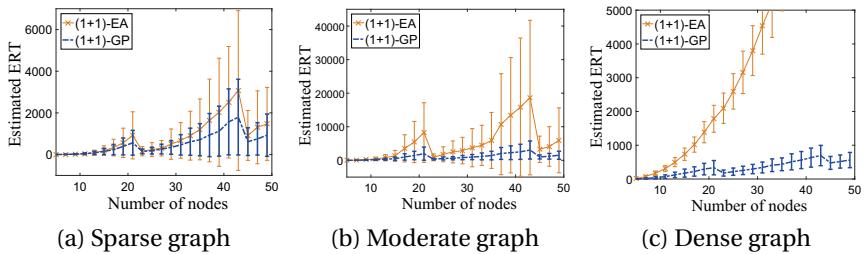


Figure 9.5. Estimated expected running time (ERT) comparison on the maximum matching problem for different graphs with $\epsilon = 0.1$.

For the minimum spanning tree problem, we also conduct experiments to compare GP with GA on sparse, moderate, and dense graphs. For each kind of graph, the range of the number n of nodes is set to $[5, 35]$. On each size of n , we use the average running time of 1,000 independent runs as an estimation of the expected running time and also record the standard deviation. For each independent run, the graph is constructed by setting the weight of each edge to be a value randomly selected from $[n]$.

The results are plotted in Figure 9.6. It can be seen that the curves of GSEMO and SMO-GP are always very close. For (1+1)-EA and (1+1)-GP, their curves are nearly overlapped in Figure 9.6(a); the curve of (1+1)-GP is much

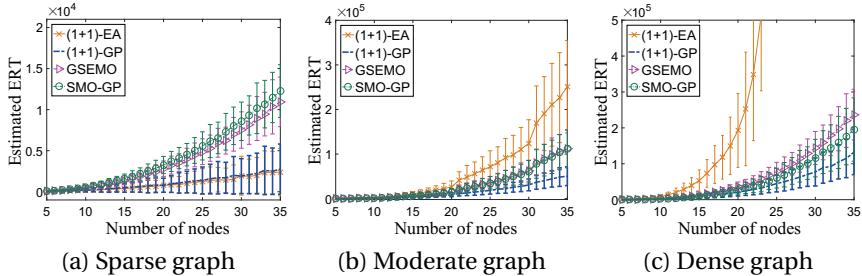


Figure 9.6. Estimated expected running time (ERT) comparison on the minimum spanning tree problem for different graphs.

lower than that of (1+1)-EA in Figures 9.6(b-c). By applying $w_{\max} \leq n$ in our experimental setting to the derived running time bounds in the third column of Table 9.1, we can get Table 9.2, from which we know that GSEMO and SMO-GP behave the same for all graphs; (1+1)-GP performs the same as (1+1)-EA for the sparse graph with $m = \Theta(n)$, but better for the moderate graph with $m = \Theta(n\sqrt{n})$ and the dense graph with $m = \Theta(n^2)$. Thus, the empirical results verify our theoretical analysis.

Table 9.2. Comparison of expected running time of GA and GP on the minimum spanning tree problem when $w_{\max} \leq n$.

Problem	(1+1)-EA	(1+1)-GP	GSEMO	SMO-GP
Minimum spanning tree	$O(m^2 \log n)$	$O(mn \log n)$	$O(mn^2)$	$O(mn^2)$

9.5 Summary

In this chapter, we analyze the influence of solution representation by comparing the running time of GP and GA on two classical combinatorial problems, i.e., the maximum matching and minimum spanning tree problems. Our theoretical analyses show that GP with a tree-structured representation achieves better performance than the previously analyzed GA with a binary-vectored representation, and the results are verified by experiments. This provides theoretical evidence to support the usefulness of rich representation in evolutionary optimization. The analysis also suggests that the variable solution structure can be helpful for evolutionary optimization when the solution complexity can be well controlled.



Inaccurate Fitness Evaluation

In practice, optimization tasks are often conducted in noisy environments, where the fitness evaluation of solutions can be inaccurate. For example, in machine learning, a prediction model is usually evaluated on a limited amount of data, where the estimated performance generally has some deviation from the true one. Actually, noisy environments may change the properties of an optimization problem, making the task more complicated. Considering that natural phenomena have been successfully processed in noisy natural environments, EAs are likely to be able to handle noise.

It was believed that noise makes optimization harder, and thus, mechanisms have been proposed to reduce the negative influence of noise [Fitzpatrick and Grefenstette, 1988, Beyer, 2000]. Representative strategies include *threshold selection* and *sampling*. With the threshold selection strategy [Markon et al., 2001, Bartz-Beielstein and Markon, 2002], EAs accept a newly generated solution only if its fitness is better than the fitness of an old solution by at least a threshold value τ , aiming at reducing the risk of accepting a bad solution. With the sampling strategy [Arnold and Beyer, 2006], EAs evaluate the fitness of a solution for k times and then use the average to approximate the true fitness, such that the fitness estimation is more robust against the noise influence.

Several empirical observations, however, have shown that noise can have a positive influence on the performance of local search algorithms [Selman et al., 1994, Hoos and Stützle, 2000, 2005], suggesting that noise does not always have a negative influence.

These previous studies are mainly empirical, though theoretical analysis is desired for a better understanding of evolutionary optimization in noisy environments. So far only a few results have been reported on running time analysis of EAs for noisy optimization. Droste [2004] analyzed (1+1)-EA on the OneMax problem in the presence of one-bit noise and showed that the maximum level of noise allowing a polynomial running time is $O(\log n/n)$, where the level of noise is characterized by the probability of noise $p_n \in [0, 1]$ and n is the problem size. The analysis has been extended to

the LeadingOnes problem and different noise models in [Gießen and Kötzing, 2016], which also proved that small population of size $\Theta(\log n)$ could make elitist EAs, e.g., $(\mu+1)$ -EA and $(1+\lambda)$ -EA, perform well under high levels of noise. The robustness of population against noise has been proved in the setting of non-elitist EAs with mutation only [Dang and Lehre, 2015] or uniform recombination only [Prügel-Bennett et al., 2015]. However, Friedrich et al. [2017] showed the limitation of population in dealing with noise by proving that $(\mu+1)$ -EA requires super-polynomial time to solve OneMax in the presence of additive Gaussian noise $\mathcal{N}(0, \sigma^2)$ with $\sigma^2 \geq n^3$. This difficulty can be overcome by the compact genetic algorithm [Friedrich et al., 2017] and a simple ACO algorithm [Friedrich et al., 2016], both finding the optimal solution in polynomial time with a high probability. There is a sequence of papers analyzing the running time of ACO on single destination shortest path problems with edge weights disturbed by noise [Sudholt and Thyssen, 2012, Doerr et al., 2012a, Feldmann and Kötzing, 2013].

Many fundamental theoretical issues on noisy evolutionary optimization remain to be addressed. In this chapter, we first present the analysis of the influence of noise on the running time of EAs [Qian et al., 2018d]. We prove that on two *EA-hard* [He and Yao, 2004] problems, i.e., Trap and Peak, noise is helpful, whereas on the *EA-easy* [He and Yao, 2004] problem, i.e., OneMax, noise is harmful. We hypothesize that the negative influence of noise decreases as the problem hardness increases, and noise can even bring a positive influence when the problem is quite hard. This hypothesis is supported by experiments on the $\text{Jump}_{m,n}$ problem and the minimum spanning tree problem, both having an adjustable hardness parameter. Then, we present the analysis of the usefulness of two commonly employed noise-handling strategies: threshold selection and sampling [Qian et al., 2018c,d]. We prove that by using proper threshold or sample size, both strategies can reduce the running time of EAs from exponential to polynomial under high levels of noise.

The rest of this chapter is organized as follows. Section 10.1 introduces the concerned noise models. Section 10.2 studies the influence of noise. The effectiveness of threshold selection and sampling is then analyzed in Sections 10.3 and 10.4, respectively. Section 10.5 presents empirical verification. Section 10.6 concludes this chapter.

10.1 Noisy Optimization

A general optimization problem can be represented as $\arg \max_s f(s)$, where the objective f is also called fitness in the context of evolutionary computation. In real-world optimization tasks, the fitness evaluation for a solution is usually disturbed by noise, and consequently we can hardly obtain the exact fitness value but only a noisy one.

Noise models can be generally divided into two categories: *prior* and *posterior* [Jin and Branke, 2005, Gießen and Kötzing, 2016]. Prior noise comes from the variation on a solution itself rather than the fitness evaluation process. One-bit noise in Definition 10.1 is a representative one, which flips a random bit of a solution before evaluation with probability p_n . It often occurs when optimizing pseudo-Boolean problems over $\{0, 1\}^n$, and has been examined in analyzing the running time of EAs in noisy optimization [Droste, 2004] and used to understand the role of noise in stochastic local search [Selman et al., 1994, Hoos and Stützle, 1999, Mengshoel, 2008].

Definition 10.1 (One-bit Noise). Given a parameter $p_n \in [0, 1]$, let $f^n(s)$ and $f(s)$ denote the noisy and true fitness of a binary solution $s \in \{0, 1\}^n$, respectively, then

$$f^n(s) = \begin{cases} f(s) & \text{with probability } 1 - p_n; \\ f(s') & \text{with probability } p_n, \end{cases} \quad (10.1)$$

where s' is generated by flipping a uniformly randomly chosen bit of s .

We also consider a variant of one-bit noise called *asymmetric* one-bit noise in Definition 10.2. Inspired from the asymmetric mutation operator [Jansen and Sudholt, 2010], asymmetric one-bit noise flips a specific bit position with probability depending on the number of bit positions that take the same value. For the flipping of asymmetric one-bit noise on a solution $s \in \{0, 1\}^n$, the probability of flipping a specific 0-bit is $(1/2) \cdot (1/|s|_0)$, and the probability of flipping a specific 1-bit is $(1/2) \cdot (1/(n - |s|_0))$. Note that for one-bit noise, the probability of flipping any specific bit is $1/n$.

Definition 10.2 (Asymmetric One-bit Noise). Given a parameter $p_n \in [0, 1]$, let $f^n(s)$ and $f(s)$ denote the noisy and true fitness of a binary solution $s \in \{0, 1\}^n$, respectively, then

$$f^n(s) = \begin{cases} f(s) & \text{with probability } 1 - p_n; \\ f(s') & \text{with probability } p_n, \end{cases} \quad (10.2)$$

where s' is generated by flipping the j -th bit of s , and j is a uniformly randomly chosen position of

$$\begin{cases} \text{all bits of } s & \text{if } |s|_0 = 0 \text{ or } n; \\ \begin{cases} 0\text{-bits of } s & \text{with probability } 1/2; \\ 1\text{-bits of } s & \text{with probability } 1/2. \end{cases} & \text{otherwise.} \end{cases} \quad (10.3)$$

Posterior noise comes from the variation on the fitness of a solution. Representative models include additive (multiplicative) noise, which adds (multiplies) a value drawn from some distribution [Beyer, 2000, Jin and Branke, 2005]. In this chapter, we consider two distributions, i.e., Gaussian distribution $\mathcal{N}(\theta, \sigma^2)$ and uniform distribution $\mathcal{U}(\delta_1, \delta_2)$.

Definition 10.3 (Additive Noise). Given some distribution \mathcal{D} , let $f^n(s)$ and $f(s)$ denote the noisy and true fitness of a solution s , respectively, then

$$f^n(s) = f(s) + \delta, \quad (10.4)$$

where δ is randomly drawn from \mathcal{D} , denoted by $\delta \sim \mathcal{D}$.

Definition 10.4 (Multiplicative Noise). Given some distribution \mathcal{D} , let $f^n(s)$ and $f(s)$ denote the noisy and true fitness of a solution s , respectively, then

$$f^n(s) = f(s) \cdot \delta, \quad (10.5)$$

where δ is randomly drawn from \mathcal{D} , denoted by $\delta \sim \mathcal{D}$.

In the following analysis, if a lemma, theorem or corollary is presented without proof, its proof is provided in Appendix A.5.

10.2 Influence of Noisy Fitness

In this section, we first prove that for the Trap and Peak problems, noise can make optimization easier for EAs, and then prove that for the One-Max problem, noise can make optimization harder for EAs. By “easier” (or “harder”), we mean that the EA with noise requires less (or more) expected running time than that without noise to find an optimal solution.

10.2.1 Noise Helpful Cases

Most practical EAs employ time-invariant operators, and thus, an EA can be modeled by a homogeneous Markov chain. In the following analysis, we always denote an EA without/with noise by $\{\xi'_t\}_{t=0}^{+\infty}/\{\xi_t\}_{t=0}^{+\infty}$, and denote the CFHT-Partition of $\{\xi'_t\}_{t=0}^{+\infty}$ in Definition 7.2 by $\{\mathcal{X}_0, \mathcal{X}_1, \dots, \mathcal{X}_m\}$.

An evolutionary process can be characterized by variation, i.e., producing new solutions, and selection, i.e., weeding out bad solutions. Denote the state spaces before and after variation by \mathcal{X} and \mathcal{X}_{var} , respectively, and then the variation process is a mapping $\mathcal{X} \rightarrow \mathcal{X}_{var}$ and the selection process is a mapping $\mathcal{X} \times \mathcal{X}_{var} \rightarrow \mathcal{X}$, e.g., for $(1+\lambda)$ -EA on any pseudo-Boolean problem, $\mathcal{X} = \{0, 1\}^n$ and $\mathcal{X}_{var} = \{\{s_1, s_2, \dots, s_\lambda\} \mid s_i \in \{0, 1\}^n\}$. Note that \mathcal{X} is just the state space of the Markov chain. $\forall x \in \mathcal{X}, x' \in \mathcal{X}_{var}$, let $P_{var}(x, x')$ denote the state transition probability by the variation process. Let \mathcal{S}^* denote the optimal solution set. As presented in Definition 10.5, a Markov chain is called *deceptive* if the chain always jumps to worse states. Note that the concerned state here may be a multiset of solutions. For two multisets $y \subseteq x$, it means that $\forall s \in y : s \in x$.

Definition 10.5 (Deceptive Markov Chain). A homogeneous Markov chain $\{\xi'_t\}_{t=0}^{+\infty}$ modeling an EA optimizing a problem without noise is deceptive, if $\forall x \in \mathcal{X}_k$ with $k \geq 1$, it holds that

$$\forall 1 \leq j \leq k-1 : P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x) = 0, \quad (10.6)$$

$$\forall k+1 \leq i \leq m :$$

$$\sum_{j=i}^m P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x) \geq \sum_{\substack{x' \cap \mathcal{S}^* = \emptyset, \\ \exists y \in \cup_{j=i}^m \mathcal{X}_j : y \subseteq x \cup x'}} P_{var}(x, x'). \quad (10.7)$$

The following theorem shows that noise can be helpful if an evolutionary process is deceptive and an optimal solution will always be accepted once generated in the noisy evolutionary process.

Theorem 10.1. For an EA \mathcal{A} optimizing a problem f , which can be modeled by a deceptive Markov chain, if

$$\forall x \notin \mathcal{X}_0 : P(\xi'_{t+1} \in \mathcal{X}_0 \mid \xi'_t = x) = \sum_{x' \cap \mathcal{S}^* \neq \emptyset} P_{var}(x, x'), \quad (10.8)$$

then noise makes f easier for \mathcal{A} .

We apply this theorem to analyze a concrete deceptive evolutionary process, i.e., $(1+\lambda)$ -EA optimizing the Trap problem. For the Trap problem in Definition 2.6, it is to maximize the number of 0-bits except for the optimal solution 1^n . The CFHT $\mathbb{E}[\tau' \mid \xi'_0 = x]$ depends only on $|x|_0$. Note that for $(1+\lambda)$ -EA, one state x just corresponds to one solution $s \in \{0, 1\}^n$. Let $\mathbb{E}_1(j)$ denote $\mathbb{E}[\tau' \mid \xi'_0 = x]$ with $|x|_0 = j$.

Lemma 10.1 gives the order of $\mathbb{E}_1(j)$, which increases with j . This result generalizes our analysis on the CFHT of $(1+1)$ -EA solving the Trap problem, i.e., Lemma 7.2.

Lemma 10.1. For any mutation probability $p \in (0, 0.5)$, it holds that

$$\mathbb{E}_1(0) < \mathbb{E}_1(1) < \mathbb{E}_1(2) < \cdots < \mathbb{E}_1(n). \quad (10.9)$$

Theorem 10.2 shows that additive and multiplicative noise with some specific uniform distributions can make $(1+\lambda)$ -EA easier to solve the Trap problem. The proof intuition is to show that the process of $(1+\lambda)$ -EA solving Trap is deceptive and then verify the condition of Theorem 10.1.

Theorem 10.2. Either additive uniform noise with $\delta_2 - \delta_1 < 2n$ or multiplicative uniform noise with $\delta_2 > \delta_1 > 0$ makes the Trap problem with $c = 3n$ easier for $(1+\lambda)$ -EA with any mutation probability less than 0.5.

Proof. First, we show that $(1+\lambda)$ -EA optimizing the Trap problem can be modeled by a deceptive Markov chain. By Lemma 10.1, the CFHT-Partition of $\{\xi'_t\}_{t=0}^{+\infty}$ is $\mathcal{X}_i = \{x \in \{0, 1\}^n \mid |x|_0 = i\} \forall 0 \leq i \leq n$ and m in Definition 7.2 is equal to n here.

$\forall x \in \mathcal{X}_k$ with $k \geq 1$, let P_0 and P_j ($1 \leq j \leq n$) denote the probability that for the λ offspring solutions x_1, \dots, x_λ generated by bit-wise mutation on x , $\min\{|x_1|_0, \dots, |x_\lambda|_0\} = 0$, i.e., the smallest number of 0-bits is 0, and $\min\{|x_1|_0, \dots, |x_\lambda|_0\} > 0 \wedge \max\{|x_1|_0, \dots, |x_\lambda|_0\} = j$, i.e., the largest number of 0-bits is j while the smallest number of 0-bits is larger than 0, respectively. For $\{\xi'_t\}_{t=0}^{+\infty}$, because only the optimal solution or the solution with the largest number of 0-bits among the parent solution and λ offspring solutions will be accepted, we have

$$\forall 1 \leq j \leq k-1 : P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x) = 0, \quad (10.10)$$

$$\forall k+1 \leq j \leq n : P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x) = P_j, \quad (10.11)$$

implying that Eqs. (10.6) and (10.7) holds.

Next, we show that Eq. (10.8), i.e., the condition of Theorem 10.1, holds. For $\{\xi_t\}_{t=0}^{+\infty}$ with additive uniform noise, considering $\delta_2 - \delta_1 < 2n$, we have

$$f^n(1^n) \geq f(1^n) + \delta_1 > 2n + \delta_2 - 2n = \delta_2, \quad (10.12)$$

$$\forall y \neq 1^n : f^n(y) \leq f(y) + \delta_2 \leq \delta_2. \quad (10.13)$$

For multiplicative uniform noise, considering $\delta_2 > \delta_1 > 0$, we have $f^n(1^n) > 0$ and $\forall y \neq 1^n : f^n(y) \leq 0$. Thus, for these two types of noise, we have $\forall y \neq 1^n : f^n(1^n) > f^n(y)$, implying that if the optimal solution 1^n is generated, it will always be accepted. As $\mathcal{X}_0 = \{1^n\}$, we have $P(\xi_{t+1} \in \mathcal{X}_0 \mid \xi_t = x) = P_0$, implying that Eq. (10.8) holds.

By Theorem 10.1, we prove that the Trap problem with $c = 3n$ becomes easier for $(1+\lambda)$ -EA under these two types of noise. \square

In addition to deceptive problems, we show that noise can also make *flat* problems easier for EAs. We take the Peak problem in Definition 2.5 for an example, which has the same fitness for all solutions except for the optimal solution 1^n . It provides no information for the search direction, and thus is hard for EAs. We analyze $(1+1)$ -EA $^\neq$ optimizing the Peak problem. As introduced in Section 2.1, $(1+1)$ -EA $^\neq$ is the same as $(1+1)$ -EA except that it employs the strict selection strategy; that is, line 4 of Algorithm 2.1 changes to “**if** $f(s') > f(s)$ **then**”. The expected running time of $(1+1)$ -EA $^\neq$ on Peak is lower bounded by $e^{n \log(n/2)}$ [Droste et al., 2002].

Theorem 10.3. *One-bit noise with $p_n \in (0, 1)$ being a constant makes the Peak problem easier for $(1+1)$ -EA $^\neq$, when starting from an initial solution with the number of 0-bits more than $(1 + p_n)/(p_n(1 - 1/n))$.*

This theorem implies that the Peak problem becomes easier under noise when starting from an initial solution s with a large number of 0-bits. From

the proof, we can see that the reason for requiring a large $|s|_0 = i$ is to make $\text{mut}_{i \rightarrow 1}$, i.e., the probability of mutating s to have one 0-bit, much larger than $\text{mut}_{i \rightarrow 0}$, i.e., the probability of mutating s to the optimal solution, enabling that the negative influence of rejecting the optimal solution by noise can be compensated by the positive influence of accepting the offspring solution with one 0-bit.

10.2.2 Noise Harmful Cases

In this section, we give a noise-harmful example rigorously. In detail, we prove that noise will make OneMax harder for $(1+\lambda)$ -EA. Let $\{\xi_t\}_{t=0}^{+\infty}/\{\xi'_t\}_{t=0}^{+\infty}$ model $(1+\lambda)$ -EA with/without noise for maximizing OneMax. The CFHT $\mathbb{E}[\tau' | \xi'_0 = x]$ depends only on $|x|_0$. Let $\mathbb{E}_2(j)$ denote $\mathbb{E}[\tau' | \xi'_0 = x]$ with $|x|_0 = j$. The order of $\mathbb{E}_2(j)$ is shown in Lemma 10.2, which generalizes our analysis on the CFHT of $(1+1)$ -EA solving the OneMax problem in Lemma 4.2.

Lemma 10.2. *For any mutation probability $p \in (0, 0.5)$, it holds that*

$$\mathbb{E}_2(0) < \mathbb{E}_2(1) < \mathbb{E}_2(2) < \cdots < \mathbb{E}_2(n). \quad (10.14)$$

Theorem 10.4. *Any noise makes the OneMax problem harder for $(1+\lambda)$ -EA with any mutation probability less than 0.5.*

Proof. We use Lemma 7.1 for the proof. By Lemma 10.2, the CFHT-Partition of $\{\xi'_t\}_{t=0}^{+\infty}$ is $\mathcal{X}_i = \{x \in \{0, 1\}^n \mid |x|_0 = i\} \forall 0 \leq i \leq n$.

For any non-optimal solution $x \in \mathcal{X}_k$ with $k > 0$, let P_j ($0 \leq j \leq n$) denote the probability for the smallest number of 0-bits of the λ offspring solutions generated by bit-wise mutation on x to be j . For $\{\xi'_t\}_{t=0}^{+\infty}$, because the solution with the smallest number of 0-bits among the parent solution and λ offspring solutions will be accepted, we have

$$\forall 0 \leq j \leq k-1 : P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x) = P_j, \quad (10.15)$$

$$P(\xi'_{t+1} \in \mathcal{X}_k \mid \xi'_t = x) = \sum_{j=k}^n P_j. \quad (10.16)$$

For $\{\xi_t\}_{t=0}^{+\infty}$, due to the fitness evaluation disturbed by noise, the solution with the smallest number of 0-bits among the parent and λ offspring solutions may be rejected. Thus, we have

$$0 \leq i \leq k-1 : \sum_{j=0}^i P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) \leq \sum_{j=0}^i P_j. \quad (10.17)$$

Combining Eqs. (10.15) to (10.17), we have

$$\forall 0 \leq i \leq n-1 : \sum_{j=0}^i P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) \leq \sum_{j=0}^i P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x), \quad (10.18)$$

implying that the condition, i.e., Eq. (7.3), of Lemma 7.1 holds. Thus, we have $\mathbb{E}[\tau | \xi_0 \sim \pi_0] \geq \mathbb{E}[\tau' | \xi'_0 \sim \pi_0]$, i.e., noise makes the OneMax problem harder for $(1+\lambda)$ -EA. \square

10.3 Denoise by Threshold Selection

During the process of evolutionary optimization, most of the improvements in one generation are small. Due to noisy fitness evaluation, a considerable portion of these improvements are not real, where a worse solution appears to have a “better” fitness and then survives to replace a better solution which has a seemingly “worse” fitness. This may mislead the search direction of EAs, and then slow down EAs or make EAs get trapped in local optima. To deal with this problem, the threshold selection strategy can be adopted [Markon et al., 2001, Bartz-Beielstein, 2005].

- **threshold selection:** an offspring solution will be accepted only if its fitness is larger than that of the parent solution by at least a predefined threshold $\tau \geq 0$.

For example, for $(1+1)$ -EA with threshold selection, line 4 of Algorithm 2.1 changes to “**if** $f(s') \geq f(s) + \tau$ **then**” rather than “**if** $f(s') \geq f(s)$ **then**”. Such a strategy can reduce the risk of accepting a bad solution due to noise. Although the good local performance, i.e., the progress of one step, of EAs with threshold selection has been shown on some problems [Markon et al., 2001, Bartz-Beielstein and Markon, 2002], its usefulness for the global performance, i.e., the running time until finding an optimal solution, of EAs under noise is not yet clear.

10.3.1 Threshold Selection

We analyze the running time of $(1+1)$ -EA with threshold selection solving OneMax under one-bit noise to see whether threshold selection is useful. Note that the analysis here assumes using re-evaluation, i.e., whenever the fitness of a solution is required, EAs make an independent evaluation of the solution regardless of whether it has been evaluated before. For $(1+1)$ -EA solving OneMax under one-bit noise, it has been known that the expected running time is polynomial iff the noise probability $p_n = O(\log n/n)$.

Theorem 10.5. [Droste, 2004] *For $(1+1)$ -EA solving OneMax under one-bit noise, the expected running time is polynomial when $p_n = O(\log n/n)$, and super-polynomial when $p_n = \omega(\log n/n)$.*

Next, we analyze the expected running time of $(1+1)$ -EA using threshold selection on the OneMax problem for different threshold values τ . Note

that the minimum fitness gap for the OneMax problem is 1. Thus, we first analyze $\tau = 1$. The following theorem shows that by using threshold selection with $\tau = 1$, (1+1)-EA can always solve OneMax in polynomial time regardless of the probability of one-bit noise.

Theorem 10.6. *For (1+1)-EA solving OneMax under one-bit noise, if using threshold selection with $\tau = 1$, the expected running time is polynomial.*

Theorem 10.6 can be derived directly from the following lemma, implying the expected running time upper bound $O(n \log n)$ for $p_n \leq 1/(\sqrt{2}e)$ and $O(n^2 \log n)$ for $p_n > 1/(\sqrt{2}e)$.

Lemma 10.3. *For (1+1)-EA using threshold selection with $\tau = 1$ on OneMax under one-bit noise, the expected running time is $O(n^2 \log n / p_n^2)$, and specifically $O(n \log n)$ when $p_n \leq 1/(\sqrt{2}e)$.*

Proof. We use additive drift analysis in Theorem 2.3 for the proof. We first construct a distance function $V(x)$ as $\forall x \in \mathcal{X} = \{0, 1\}^n : V(x) = H_{|x|_0}$. It can be verified that $V(x \in \mathcal{X}^* = \{1^n\}) = 0$ and $V(x \notin \mathcal{X}^*) > 0$.

We examine $\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) | \xi_t = x] \forall x$ with $V(x) > 0$, i.e., $x \notin \mathcal{X}^*$. Let i denote the number of 0-bits of the current solution x , where $1 \leq i \leq n$. Let $p_{i,i+d}$ be the probability for the next solution after mutation and selection to have $(i+d)$ number of 0-bits, where $-i \leq d \leq n-i$. We have

$$\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) | \xi_t = x] = H_i - \sum_{d=-i}^{n-i} p_{i,i+d} \cdot H_{i+d}. \quad (10.19)$$

Next, we analyze $p_{i,i+d}$ for $1 \leq i \leq n$. Let P_d denote the probability for the offspring solution x' by bit-wise mutation on x to have $(i+d)$ number of 0-bits, where $-i \leq d \leq n-i$. Note that one-bit noise can change the true fitness of a solution by at most 1, i.e., $|f^n(x) - f(x)| \leq 1$.

- (1) When $d \geq 2$, $f^n(x') \leq n - i - d + 1 \leq n - i - 1 \leq f^n(x)$. Because an offspring solution will be accepted only if $f^n(x') \geq f^n(x) + 1$, the offspring solution x' will be discarded in this case, implying $\forall d \geq 2 : p_{i,i+d} = 0$.
- (2) When $d = 1$, the offspring solution x' will be accepted only if $f^n(x') = n - i \wedge f^n(x) = n - i - 1$, the probability of which is $(p_n(i+1)/n) \cdot (p_n(n-i)/n)$, as it requires to flip one 0-bit of x' and one 1-bit of x . Thus,

$$p_{i,i+1} = P_1 \cdot \frac{p_n(i+1)}{n} \frac{p_n(n-i)}{n}. \quad (10.20)$$

- (3) When $d = -1$, if $f^n(x) = n - i - 1$, the probability of which is $p_n(n-i)/n$, the offspring solution x' will be accepted, as $f^n(x') \geq n - i + 1 - 1 = n - i > f^n(x)$; if $f^n(x) = n - i \wedge f^n(x') \geq n - i + 1$, the probability of which is $(1-p_n) \cdot (1-p_n + p_n(i-1)/n)$, x' will be accepted; if $f^n(x) = n - i + 1 \wedge f^n(x') = n - i + 2$, the probability of which is $(p_n i/n) \cdot (p_n(i-1)/n)$, x' will be accepted; otherwise, x' will be discarded. Thus,

$$p_{i,i-1} = P_{-1} \cdot \left(\frac{p_n(n-i)}{n} + (1-p_n) \left(1 - p_n + \frac{p_n(i-1)}{n} \right) + \frac{p_n i}{n} \frac{p_n(i-1)}{n} \right). \quad (10.21)$$

(4) When $d \leq -2$, we have $p_{i,i+d} > 0$.

By applying these probabilities to Eq. (10.19), we have

$$\begin{aligned} & \mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] \\ & \geq H_i - p_{i,i-1} H_{i-1} - p_{i,i+1} H_{i+1} - (1 - p_{i,i-1} - p_{i,i+1}) H_i \\ & = p_{i,i-1} \cdot \frac{1}{i} - p_{i,i+1} \cdot \frac{1}{i+1} \\ & \geq P_{-1} \left(p_n \frac{n-i}{n} + p_n^2 \frac{i(i-1)}{n^2} \right) \frac{1}{i} - P_1 p_n^2 \frac{(i+1)(n-i)}{n^2} \frac{1}{i+1}. \end{aligned} \quad (10.22)$$

We then bound the two mutation probabilities P_{-1} and P_1 . For decreasing the number of 0-bits by 1 in mutation, it is sufficient to flip one 0-bit and keep the other bits unchanged, and thus, $P_{-1} \geq (i/n)(1 - 1/n)^{n-1}$. For increasing the number of 0-bits by 1, it requires to flip one more 1-bit than the number of 0-bits it flips, and thus,

$$\begin{aligned} P_1 &= \sum_{k=1}^{\min\{n-i,i+1\}} \binom{n-i}{k} \binom{i}{k-1} \frac{1}{n^{2k-1}} \left(1 - \frac{1}{n}\right)^{n-2k+1} \\ &\leq \frac{n-i}{n} \left(1 - \frac{1}{n}\right)^{n-1} + \sum_{k=2}^{\min\{n-i,i+1\}} \frac{1}{k!(k-1)!} \frac{(n-i)^k}{n^k} \frac{i^{k-1}}{n^{k-1}} \left(1 - \frac{1}{n}\right)^{n-2k+1} \\ &\leq \frac{n-i}{n} \left(1 - \frac{1}{n}\right)^{n-1} + \frac{i}{n} \cdot \sum_{k=2}^{\min\{n-i,i+1\}} \frac{1}{k!(k-1)!} \left(1 - \frac{1}{n}\right)^{n-1} \\ &\leq \frac{n-i}{n} \left(1 - \frac{1}{n}\right)^{n-1} + \frac{i}{n} \cdot \sum_{k=2}^{+\infty} \frac{1}{k!} \left(1 - \frac{1}{n}\right)^{n-1} \\ &= \frac{n-i}{n} \left(1 - \frac{1}{n}\right)^{n-1} + (e-2) \frac{i}{n} \left(1 - \frac{1}{n}\right)^{n-1}. \end{aligned} \quad (10.23)$$

By applying these two bounds of P_{-1} and P_1 to Eq. (10.22), we have

$$\begin{aligned} \mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] &\geq \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} p_n^2 \left(\frac{n-i}{n} + \frac{i(i-1)}{n^2} \right) \\ &\quad - \left(\frac{n-i}{n} + (e-2) \frac{i}{n} \right) \left(1 - \frac{1}{n}\right)^{n-1} p_n^2 \frac{n-i}{n^2} \\ &\geq (3-e) \frac{i}{n^2} \left(1 - \frac{1}{n}\right)^n p_n^2 \geq \frac{3-e}{2e} \frac{p_n^2}{n^2}, \end{aligned} \quad (10.24)$$

where Eq. (10.24) holds by $i \geq 1$ and $(1 - 1/n)^n \geq 1/(2e)$. Thus, by Theorem 2.3 and considering $V(x) \leq H_n < 1 + \log n$, we have

$$\mathbb{E}[\tau \mid \xi_0] \leq \frac{2e}{3-e} \frac{n^2}{p_n^2} V(\xi_0) = O\left(\frac{n^2 \log n}{p_n^2}\right), \quad (10.25)$$

i.e., the expected running time of (1+1)-EA with $\tau = 1$ on the OneMax problem is upper bounded by $O(n^2 \log n / p_n^2)$.

For $p_n \leq 1/(\sqrt{2}e)$, we can derive a tighter upper bound $O(n \log n)$ by applying proper bounds of the two probabilities $p_{i,i-1}$ and $p_{i,i+1}$ to Eq. (10.22). From cases (3) and (2) in the analysis of $p_{i,i+d}$, we have

$$p_{i,i-1} \geq P_{-1}(1 - p_n)^2 \geq \frac{i}{n} \left(1 - \frac{1}{n}\right)^{n-1} (1 - p_n)^2, \quad (10.26)$$

$$p_{i,i+1} = P_1 p_n^2 \frac{(i+1)(n-i)}{n^2} \leq \frac{(i+1)(n-i)^2}{n^3} p_n^2, \quad (10.27)$$

where Eq. (10.27) holds by $P_1 \leq (n-i)/n$ because it is necessary to flip at least one 1-bit. Then, Eq. (10.22) becomes

$$\begin{aligned} \mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] &\geq \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} (1 - p_n)^2 - \frac{(n-i)^2}{n^3} p_n^2 \\ &\geq \frac{1}{n} \left(\frac{1}{e} (1 - p_n)^2 - p_n^2\right) > 0.13 \cdot \frac{1}{n}, \end{aligned} \quad (10.28)$$

where Eq. (10.28) holds because $(1 - p_n)^2/e - p_n^2$ decreases with p_n for $p_n \leq 1/(\sqrt{2}e)$. Thus, by Theorem 2.3, we have

$$\mathbb{E}[\tau \mid \xi_0] \leq \frac{n}{0.13} V(\xi_0) = O(n \log n), \quad (10.29)$$

i.e., the expected running time of (1+1)-EA with $\tau = 1$ on the OneMax problem is upper bounded by $O(n \log n)$ for $p_n \leq 1/(\sqrt{2}e)$. \square

Next, we analyze a relatively large threshold value $\tau = 2$. In the following analysis, let $\text{poly}(n)$ indicate any polynomial of n .

Theorem 10.7. *For (1+1)-EA solving OneMax under one-bit noise, if using threshold selection with $\tau = 2$, the expected running time is polynomial iff $p_n = (1/O(\text{poly}(n))) \cap (1 - 1/O(\text{poly}(n)))$.*

Theorem 10.7 can be derived by the following two lemmas. Lemma 10.4 shows the expected running time upper bound $O(n \log n / (p_n(1 - p_n)))$, implying that the expected running time is polynomial if $1/(p_n(1 - p_n)) = O(\text{poly}(n))$. Lemma 10.5 shows the lower bound $\Omega(n \log n + n / (p_n(1 - p_n)))$, implying that the expected running time is super-polynomial if $1/(p_n(1 - p_n)) = \omega(\text{poly}(n))$. Overall, the expected running time is polynomial iff $1/(p_n(1 - p_n)) = O(\text{poly}(n))$, i.e., $p_n = (1/O(\text{poly}(n))) \cap (1 - 1/O(\text{poly}(n)))$.

Lemma 10.4. *For (1+1)-EA using threshold selection with $\tau = 2$ on OneMax under one-bit noise, the expected running time is $O(n \log n / (p_n(1 - p_n)))$.*

Lemma 10.5. *For (1+1)-EA using threshold selection with $\tau = 2$ on OneMax under one-bit noise, the expected running time is $\Omega(n \log n + n/(p_n(1 - p_n)))$.*

For larger threshold values $\tau > 2$, we have:

Theorem 10.8. *For (1+1)-EA solving OneMax under one-bit noise, if using threshold selection with $\tau > 2$, the expected running time is infinite.*

Theorems 10.7 and 10.8 show that using threshold selection with improper threshold τ can reduce the noise-tolerant ability, and even bring a negative influence. Thus, it is important to select a proper τ value.

10.3.2 Smooth Threshold Selection

In this section, we will show the limitation of threshold selection, and further present an improved version, called *smooth threshold selection*.

Theorem 10.9. *For (1+1)-EA solving OneMax under asymmetric one-bit noise, if using threshold selection with $\tau \geq 0$, the expected running time is at least exponential for $p_n = 1$.*

Theorem 10.9 discloses that threshold selection can be ineffective. In the concerned situation, when $0 \leq \tau \leq 1$, it has a large probability of accepting false progress, leading to a negative drift and thus the exponential running time; when $\tau \geq 2$, although the probability of accepting false progress is 0, i.e., $\forall d \geq 1 : p_{i,i+d} = 0$, it has a small probability of accepting true progress, i.e., $p_{1,0} = 0$, leading to the infinite running time; setting τ between 1 and 2 is useless, because the minimum fitness gap is 1, making a value of $\tau \in (1, 2)$ equivalent to $\tau = 2$.

We present the smooth threshold selection strategy in Definition 10.6, which modifies the original threshold selection strategy by changing the hard threshold value to a smooth one. The “smooth” means that the offspring solution will be accepted with some probability, when the fitness gap between the offspring and the parent satisfies the threshold. For example, (1)+(0.1)-smooth threshold selection accepts the offspring solution with probability 0.9 when the fitness gap is 1; this makes a fractional threshold 1.1 effective. Such a strategy of accepting new solutions probabilistically based on the fitness is somewhat similar to the acceptance strategy of *simulated annealing* [Kirkpatrick, 1984].

Definition 10.6 (Smooth Threshold Selection). *Let δ be the gap between the fitness of the offspring solution s' and the parent solution s , i.e., $\delta = f(s') - f(s)$. Given a threshold $(a) + (b)$ with $b \in [0, 1]$, the selection process will behave as follows:*

- (1) *if $\delta < a$, s' will be rejected;*
- (2) *if $\delta = a$, s' will be accepted with probability $1 - b$;*
- (3) *if $\delta > a$, s' will be accepted.*

Theorem 10.10 shows that using smooth threshold selection with proper threshold values can make (1+1)-EA solve OneMax in polynomial time regardless of the probability of asymmetric one-bit noise.

Theorem 10.10. *For (1+1)-EA solving OneMax under asymmetric one-bit noise, if using $(1) + (1 - 1/(2en))$ -smooth threshold selection, the expected running time is polynomial.*

We draw an intuitive understanding from the proof of Theorem 10.10 of why smooth threshold selection can be better than the original one. By changing the hard threshold to smooth, it can not only make the probability of accepting a false better solution in one step small enough, i.e., $p_{i,i-1} \geq p_{i,i+1}$, but also make the probability of producing real progress in one step large enough, i.e., $p_{i,i-1}$ not small.

10.4 Denoise by Sampling

In noisy evolutionary optimization, sampling in Definition 10.7 has often been used to reduce the negative influence of noise [Aizawa and Wah, 1994, Stagge, 1998, Branke and Schmidt, 2003, 2004]. It, instead of evaluating the fitness of one solution only once, evaluates the fitness for k times and then uses the average to approximate the true fitness. Sampling can reduce the standard deviation of noise by a factor of \sqrt{k} , while increasing the computational cost k times. This makes the fitness estimation more robust against noise, but computationally more expensive. In this section, we theoretically study the effectiveness of sampling. We prove that using sampling can speed up the running time exponentially for (1+1)-EA solving OneMax and LeadingOnes under both prior and posterior noise.

Definition 10.7 (Sampling). *Sampling evaluates the fitness of a solution for k times independently and obtains the noisy fitness values $f_1^n(s), f_2^n(s), \dots, f_k^n(s)$, and then outputs their average as*

$$\hat{f}(s) = \frac{1}{k} \sum_{i=1}^k f_i^n(s). \quad (10.30)$$

Akimoto et al. [2015] showed that using sampling with a large enough k can make optimization under additive unbiased noise behave as the optimization in a noise-free environment, and thus concluded that noisy optimization using sampling could be solved in $k \cdot r$ running time, where r is the noise-free running time. This result, however, does not describe the influence of sampling on the running time, because the running time in noisy optimization without sampling was not compared.

10.4.1 Robustness against Prior Noise

By comparing the expected running time of (1+1)-EA with/without sampling for solving OneMax and LeadingOnes under one-bit noise, we first show the robustness of sampling against prior noise. For (1+1)-EA using sampling, line 4 of Algorithm 2.1 changes to “**if** $\hat{f}(s') \geq \hat{f}(s)$ **then**”.

For the OneMax problem, one-bit noise with $p_n = 1$ is considered. We first analyze the case in which sampling is not used. Note that the expected running time is super-polynomial for $p_n = \omega(\log n/n)$, as shown in Theorem 10.5. Gießen and Kötzing [2016] re-proved the super-polynomial lower bound for $p_n = \omega(\log n/n) \cap (1 - \omega(\log n/n))$ by using the simplified negative drift analysis approach in Theorem 2.5. However, their proof does not cover $p_n = 1$. Here, we use the simplified negative drift analysis approach with self-loops in Theorem 2.6 to prove the lower bound of the exponential running time for $p_n = 1$, as shown in Theorem 10.11.

Theorem 10.11. *For (1+1)-EA solving OneMax under one-bit noise with $p_n = 1$, the expected running time is exponential.*

Next we analyze the case where sampling with $k = 2$ is used. The running time is still exponential, as shown in Theorem 10.12. The proof is similar to that of Theorem 10.11. The change of the transition probabilities, i.e., $p_{i,i+d}$, led by increasing k from 1 to 2 does not affect the application of the simplified negative drift analysis approach with self-loops.

Theorem 10.12. *For (1+1)-EA solving OneMax under one-bit noise with $p_n = 1$, if using sampling with $k = 2$, the expected running time is exponential.*

We have shown that sampling with $k = 2$ is ineffective. In the following, we prove that increasing k from 2 to 3 can reduce the expected running time to a polynomial as shown in Theorem 10.13, the proof of which is accomplished by applying multiplicative drift analysis in Theorem 2.4.

Theorem 10.13. *For (1+1)-EA solving OneMax under one-bit noise with $p_n = 1$, if using sampling with $k = 3$, the expected running time is $O(n \log n)$.*

Proof. We use Theorem 2.4 for the proof. We first construct a distance function $V(x)$ as $\forall x \in \mathcal{X} = \{0, 1\}^n : V(x) = |x|_0$. It can be verified that $V(x \in \mathcal{X}^*) = \{1^n\} = 0$ and $V(x \notin \mathcal{X}^*) > 0$.

We examine $\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] \forall x$ with $V(x) > 0$, i.e., $x \notin \mathcal{X}^*$. Let i denote the number of 0-bits of the current solution, where $1 \leq i \leq n$. Let $p_{i,i+d}$ be the probability for the next solution after mutation and selection to have $(i + d)$ number of 0-bits, where $-i \leq d \leq n - i$. Note that we are referring to the true number of 0-bits of a solution rather than the effective number of 0-bits after noisy evaluation. Thus, we have

$$\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] = \sum_{d=1}^i d \cdot p_{i,i-d} - \sum_{d=1}^{n-i} d \cdot p_{i,i+d}. \quad (10.31)$$

Next we analyze $p_{i,i+d}$ for $1 \leq i \leq n$ as in the proof of Theorem 10.11. Note that for a solution x , the fitness value output by sampling with $k = 3$ is the average of noisy fitness values output by three independent fitness evaluations, i.e., $\hat{f}(x) = (f_1^n(x) + f_2^n(x) + f_3^n(x))/3$.

- (1) When $d \geq 3$, $\hat{f}(x') \leq n - i - d + 1 \leq n - i - 2 < \hat{f}(x)$. Thus, the offspring x' will be discarded, and we have $\forall d \geq 3 : p_{i,i+d} = 0$.
- (2) When $d = 2$, x' will be accepted iff $\hat{f}(x') = n - i - 1 = \hat{f}(x)$, the probability of which is $((i+2)/n)^3 \cdot ((n-i)/n)^3$, as it requires to flip one 0-bit of x' and one 1-bit of x in all three noisy fitness evaluations. Thus,

$$p_{i,i+2} = P_2 \cdot \left(\frac{i+2}{n} \right)^3 \left(\frac{n-i}{n} \right)^3. \quad (10.32)$$

- (3) When $d = 1$, there are three possible cases for the acceptance of x' : $\hat{f}(x') = n - i \wedge \hat{f}(x) = n - i - 1$, $\hat{f}(x') = n - i \wedge \hat{f}(x) = n - i - 1/3$ and $\hat{f}(x') = n - i - 2/3 \wedge \hat{f}(x) = n - i - 1$. The probability of $\hat{f}(x') = n - i$ is $((i+1)/n)^3$, as it requires to flip one 0-bit of x in all three noisy evaluations. The probability of $\hat{f}(x') = n - i - 2/3$ is $3((i+1)/n)^2((n-i-1)/n)$, as it requires to flip one 0-bit of x in two noisy evaluations and one 1-bit in the other noisy evaluation. Similarly, we can derive that the probabilities of $\hat{f}(x) = n - i - 1$ and $\hat{f}(x) = n - i - 1/3$ are $((n-i)/n)^3$ and $3((n-i)/n)^2(i/n)$, respectively. Thus,

$$\begin{aligned} p_{i,i+1} = P_1 \cdot & \left(\left(\frac{i+1}{n} \right)^3 \left(\left(\frac{n-i}{n} \right)^3 + 3 \left(\frac{n-i}{n} \right)^2 \frac{i}{n} \right) \right. \\ & \left. + 3 \left(\frac{i+1}{n} \right)^2 \frac{n-i-1}{n} \left(\frac{n-i}{n} \right)^3 \right). \end{aligned} \quad (10.33)$$

- (4) When $d = -1$, there are three possible cases for the rejection of x' : $\hat{f}(x') = n - i \wedge \hat{f}(x) = n - i + 1$, $\hat{f}(x') = n - i \wedge \hat{f}(x) = n - i + 1/3$ and $\hat{f}(x') = n - i + 2/3 \wedge \hat{f}(x) = n - i + 1$. The probability of $\hat{f}(x') = n - i$ is $((n-i+1)/n)^3$, as it requires to flip one 1-bit of x' in all three noisy evaluations. The probability of $\hat{f}(x') = n - i + 2/3$ is $3((n-i+1)/n)^2((i-1)/n)$, as it requires to flip one 1-bit of x' in two noisy evaluations and one 0-bit in the other evaluation. Similarly, we can derive that the probabilities of $\hat{f}(x) = n - i + 1$ and $\hat{f}(x) = n - i + 1/3$ are $(i/n)^3$ and $3(i/n)^2((n-i)/n)$, respectively. Thus,

$$\begin{aligned} p_{i,i-1} = P_{-1} \cdot & \left(1 - \left(\frac{n-i+1}{n} \right)^3 \left(\left(\frac{i}{n} \right)^3 + 3 \left(\frac{i}{n} \right)^2 \frac{n-i}{n} \right) \right. \\ & \left. - 3 \left(\frac{n-i+1}{n} \right)^2 \frac{i-1}{n} \left(\frac{i}{n} \right)^3 \right). \end{aligned} \quad (10.34)$$

(5) When $d \leq -2$, $\hat{f}(x') \geq n - i - d - 1 \geq n - i + 1 \geq \hat{f}(x)$. Thus, x' will always be accepted, and we have

$$\forall d \leq -2 : p_{i,i+d} = P_d. \quad (10.35)$$

By applying these probabilities to Eq. (10.31), we have

$$\begin{aligned} & \mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] \geq p_{i,i-1} - p_{i,i+1} - 2 \cdot p_{i,i+2} \\ &= P_{-1} \cdot \left(1 - \left(\frac{n-i+1}{n} \right)^3 \left(\left(\frac{i}{n} \right)^3 + 3 \left(\frac{i}{n} \right)^2 \frac{n-i}{n} \right) - 3 \left(\frac{n-i+1}{n} \right)^2 \frac{i-1}{n} \left(\frac{i}{n} \right)^3 \right) \\ &\quad - P_1 \cdot \left(\left(\frac{i+1}{n} \right)^3 \left(\left(\frac{n-i}{n} \right)^3 + 3 \left(\frac{n-i}{n} \right)^2 \frac{i}{n} \right) + 3 \left(\frac{i+1}{n} \right)^2 \frac{n-i-1}{n} \left(\frac{n-i}{n} \right)^3 \right) \\ &\quad - 2 \cdot P_2 \cdot \left(\frac{i+2}{n} \right)^3 \left(\frac{n-i}{n} \right)^3. \end{aligned} \quad (10.36)$$

Eq. (10.36) can be simplified by simple mathematical calculations.

$$\begin{aligned} & \left(\frac{i+1}{n} \right)^3 \left(\left(\frac{n-i}{n} \right)^3 + 3 \left(\frac{n-i}{n} \right)^2 \frac{i}{n} \right) + 3 \left(\frac{i+1}{n} \right)^2 \frac{n-i-1}{n} \left(\frac{n-i}{n} \right)^3 \\ &= \frac{i+1}{n} \frac{n-i}{n} \cdot \left(-5 \left(\frac{i+1}{n} \frac{n-i}{n} \right)^2 + 3 \left(1 + \frac{1}{n} \right) \cdot \frac{i+1}{n} \frac{n-i}{n} \right) \\ &\leq \frac{9}{20} \frac{i+1}{n} \frac{n-i}{n} \left(\frac{n+1}{n} \right)^2, \end{aligned} \quad (10.37)$$

where Eq. (10.37) holds because $-5x^2 + 3(1 + 1/n)x \leq (9/20)((n+1)/n)^2$. By replacing i with $(n - i)$ in Eq. (10.37), we have

$$\begin{aligned} & \left(\frac{n-i+1}{n} \right)^3 \left(\left(\frac{i}{n} \right)^3 + 3 \left(\frac{i}{n} \right)^2 \frac{n-i}{n} \right) + 3 \left(\frac{n-i+1}{n} \right)^2 \frac{i-1}{n} \left(\frac{i}{n} \right)^3 \\ &\leq \frac{9}{20} \frac{i}{n} \frac{n-i+1}{n} \left(\frac{n+1}{n} \right)^2. \end{aligned} \quad (10.38)$$

Thus, Eq. (10.36) becomes

$$\begin{aligned} & \mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] \\ &\geq P_{-1} \cdot \left(1 - \frac{9}{20} \frac{i}{n} \frac{n-i+1}{n} \left(\frac{n+1}{n} \right)^2 \right) - P_1 \cdot \frac{9}{20} \frac{i+1}{n} \frac{n-i}{n} \left(\frac{n+1}{n} \right)^2 \\ &\quad - 2 \cdot P_2 \cdot \left(\frac{i+2}{n} \right)^3 \left(\frac{n-i}{n} \right)^3. \end{aligned} \quad (10.39)$$

We then bound the three mutation probabilities P_{-1} , P_1 and P_2 . For decreasing the number of 0-bits by 1 in mutation, it is sufficient to flip one

0-bit and keep the other bits unchanged, and thus $P_{-1} \geq (i/n)(1 - 1/n)^{n-1}$. For increasing the number of 0-bits by 2, it is necessary to flip at least two 1-bits, and thus $P_2 \leq \binom{n-i}{2}(1/n^2) = (n-i)(n-i-1)/(2n^2)$. For increasing the number of 0-bits by 1, it requires to flip one more 1-bit than the number of 0-bits it flips. As in the proof of Lemma 10.3, Eq. (10.23) shows $P_1 \leq ((n-i)/n)(1 - 1/n)^{n-1} + (e-2)(i/n)(1 - 1/n)^{n-1}$.

By applying these probability bounds to Eq. (10.39), we have

$$\begin{aligned} \mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] &\geq \frac{i}{n} \left(1 - \frac{1}{n}\right)^{n-1} \\ &\cdot \left(1 - \frac{9}{20} \left(\frac{n+1}{n}\right)^2 \left(\frac{i}{n} \frac{n-i+1}{n} + \frac{i+1}{i} \frac{n-i}{n} \left(\frac{n-i}{n} + (e-2)\frac{i}{n}\right)\right)\right) \\ &- \frac{i}{n} \cdot 2 \cdot \frac{(n-i)(n-i-1)}{2n^2} \cdot \frac{i+2}{i} \left(\frac{i+2}{n}\right)^2 \left(\frac{n-i}{n}\right)^3. \end{aligned} \quad (10.40)$$

When $i \geq 2$, $1 + 1/i \leq 3/2$ and $1 + 2/i \leq 2$. Thus, we have

$$\begin{aligned} \mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] &\geq \frac{i}{en} \left(1 - \frac{9}{20} \left(\frac{n+1}{n}\right)^2 \cdot \frac{3}{2}\right) - \frac{32}{729} \frac{i}{n} \left(\frac{n+2}{n}\right)^5 \end{aligned} \quad (10.41)$$

$$= \frac{i}{n} \cdot \left(\frac{1}{e} - \frac{27}{40e} \left(\frac{n+1}{n}\right)^2 - \frac{32}{729} \left(\frac{n+2}{n}\right)^5\right) \geq 0.003 \cdot \frac{i}{n}. \quad (10.42)$$

Note that Eq. (10.41) can be derived by

$$\begin{aligned} &\frac{(n-i)(n-i-1)}{n^2} \left(\frac{i+2}{n}\right)^2 \left(\frac{n-i}{n}\right)^3 \\ &\leq \frac{n-2}{n} \left(\frac{i+2}{n} \left(\frac{n-i}{n}\right)^2\right)^2 \leq \frac{n-2}{n} \left(\frac{4}{27} \left(\frac{n+2}{n}\right)^3\right)^2 \leq \frac{16}{729} \left(\frac{n+2}{n}\right)^5, \end{aligned} \quad (10.43)$$

and Eq. (10.42) holds with $n \geq 15$.

When $i = 1$, using Eq. (10.36), we have

$$\begin{aligned} \mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] &= P_{-1} \cdot \left(1 - \left(\left(\frac{1}{n}\right)^3 + 3\left(\frac{1}{n}\right)^2 \frac{n-1}{n}\right)\right) - 2 \cdot P_2 \cdot \left(\frac{3}{n}\right)^3 \left(\frac{n-1}{n}\right)^3 \\ &- P_1 \cdot \left(\left(\frac{2}{n}\right)^3 \left(\left(\frac{n-1}{n}\right)^3 + 3\left(\frac{n-1}{n}\right)^2 \frac{1}{n}\right) + 3\left(\frac{2}{n}\right)^2 \frac{n-2}{n} \left(\frac{n-1}{n}\right)^3\right) \\ &\geq \frac{1}{en} \left(1 - \frac{3}{n^2} - \frac{16}{n^2} - \frac{12}{n}\right) - \frac{27}{n^3} \geq 0.01 \cdot \frac{1}{n}, \end{aligned} \quad (10.44)$$

where Eq. (10.44) holds with $n \geq 18$. Thus, the condition of Theorem 2.4 holds with $\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] \geq (0.003/n) \cdot V(\xi_t)$. Considering $V_{\min} = 1$ and $V(x) \leq n$, we have

$$\mathbb{E}[\tau \mid \xi_0] \leq \frac{n}{0.003}(1 + \log V(\xi_0)) = O(n \log n), \quad (10.45)$$

implying that the expected running time is upper bounded by $O(n \log n)$. \square

Thus, we have shown that sampling is robust against noise for (1+1)-EA solving the OneMax problem in the presence of one-bit noise. By comparing Theorems 10.12 and 10.13, we also find that adding one to the value of k can lead to an exponential difference on the expected running time, disclosing that a careful selection of k is important for the effectiveness of sampling. The complexity transition from $k = 2$ to $k = 3$ is because sampling with $k = 3$ can make false progress, i.e., accepting solutions with more 0-bits, dominated by true progress, i.e., accepting solutions with less 0-bits, whereas sampling with $k = 2$ is insufficient.

For the LeadingOnes problem, one-bit noise with $p_n = 1/2$ is considered. For the case in which sampling is not used, Gießen and Kötzing [2016] have proved the exponential running time lower bound as shown in Theorem 10.14. We prove in Theorem 10.15 that sampling can reduce the expected running time to a polynomial.

Theorem 10.14. [Gießen and Kötzing, 2016] *For (1+1)-EA solving LeadingOnes under one-bit noise with $p_n = 1/2$, the expected running time is $2^{\Omega(n)}$.*

Theorem 10.15. *For (1+1)-EA solving LeadingOnes under one-bit noise with $p_n = 1/2$, if using sampling with $k = 10n^4$, the expected running time is $O(n^6)$.*

10.4.2 Robustness against Posterior Noise

By comparing the expected running time of (1+1)-EA with/without sampling for solving OneMax and LeadingOnes under additive Gaussian noise, we will prove that sampling can also be robust against posterior noise.

For the OneMax problem, additive Gaussian noise with $\sigma^2 \geq 1$ is considered. We first analyze the case in which sampling is not used. By applying the simplified negative drift analysis approach in Theorem 2.5, we prove that the expected running time is exponential.

Theorem 10.16. *For (1+1)-EA solving OneMax under additive Gaussian noise with $\sigma^2 \geq 1$, the expected running time is exponential.*

Friedrich et al. [2017] have proved that for solving OneMax under additive Gaussian noise with $\theta = 0$ and $\sigma^2 \geq n^3$, $(\mu+1)$ -EA requires superpolynomial expected running time. Our result in Theorem 10.16 is complementary to their result with $\mu = 1$, as it covers a constant variance. We

then prove in Corollary 10.1 that using sampling can reduce the expected running time to a polynomial. The intuition is that sampling with a large enough k can reduce noise to $\sigma^2 = O(\log n/n)$, allowing a polynomial running time, as shown in the following lemma.

Lemma 10.6. [Gießen and Kötzing, 2016] Suppose additive noise, sampling from some distribution \mathcal{D} with variance σ^2 . (1+1)-EA optimizes OneMax in polynomial time if $\sigma^2 = O(\log n/n)$.

Corollary 10.1. For (1+1)-EA solving OneMax under additive Gaussian noise with $\sigma^2 \geq 1$ and $\sigma^2 = O(\text{poly}(n))$, if using sampling with $k = \lceil n\sigma^2 / \log n \rceil$, the expected running time is polynomial.

Proof. The noisy fitness is $f^n(x) = f(x) + \delta$, where $\delta \sim \mathcal{N}(\theta, \sigma^2)$. The fitness output by sampling is $\hat{f}(x) = (\sum_{i=1}^k f_i^n(x))/k = (\sum_{i=1}^k f(x) + \delta_i)/k = f(x) + \sum_{i=1}^k \delta_i/k$, where $\delta_i \sim \mathcal{N}(\theta, \sigma^2)$. Thus, $\hat{f}(x) = f(x) + \delta'$, where $\delta' \sim \mathcal{N}(\theta, \sigma^2/k)$. In other words, sampling reduces the variance σ^2 of noise to σ^2/k . Because $k = \lceil n\sigma^2 / \log n \rceil$, we have $\sigma^2/k \leq \log n/n$. By Lemma 10.6, the expected number of iterations of (1+1)-EA for finding the optimal solution is polynomial. The expected running time is $2k$ times the expected number of iterations. Considering $\sigma^2 = O(\text{poly}(n))$, the expected running time is polynomial. \square

For the LeadingOnes problem, additive Gaussian noise with $\sigma^2 \geq n^2$ is considered. We first analyze the case where sampling is not used. Using the simplified negative drift analysis approach in Theorem 2.5, we prove that the expected running time is exponential.

Theorem 10.17. For (1+1)-EA solving LeadingOnes under additive Gaussian noise with $\sigma^2 \geq n^2$, the expected running time is exponential.

Next we prove in Corollary 10.2 that using sampling can reduce the expected running time to a polynomial. The intuition is that sampling with a large enough k can reduce noise to $\sigma^2 \leq 1/(12en^2)$, allowing a polynomial running time, as shown in the following lemma.

Lemma 10.7. [Gießen and Kötzing, 2016] Suppose additive noise, sampling from some distribution \mathcal{D} with variance σ^2 . (1+1)-EA optimizes LeadingOnes in $O(n^2)$ time if $\sigma^2 \leq 1/(12en^2)$.

Corollary 10.2. For (1+1)-EA solving LeadingOnes under additive Gaussian noise with $\sigma^2 \geq n^2$ and $\sigma^2 = O(\text{poly}(n))$, if using sampling with $k = \lceil 12en^2\sigma^2 \rceil$, the expected running time is polynomial.

Akimoto et al. [2015] have studied the running time of a general optimization algorithm solving a class of integer-valued functions under additive Gaussian noise. They have proved in Theorem 2.2 of [Akimoto et al.,

2015] that with a high probability, the running time by using sampling is no greater than the noise-free running time by a logarithmic factor. Note that although their algorithm and problem setting covers the case in our consideration when (1+1)-EA is used to solve OneMax and LeadingOnes, the employed sampling techniques are different: a rounding function $\lfloor \hat{f} + 0.5 \rfloor$ is applied to the average fitness value \hat{f} in [Akimoto et al., 2015], whereas the average value is used directly in our analysis.

10.5 Empirical Verification

In this section, we will verify our theoretical analyses on the influence of noise by experiments. For any given configuration, we will run the EA for 1,000 times independently. In each run, we record the number of fitness evaluations until an optimal solution is found. The running time values of the 1,000 runs are averaged for an estimation of the expected running time.

10.5.1 Noise Helpful Cases

We first present the experimental results on the Trap and Peak problems to verify Theorems 10.2 and 10.3.

As for Theorem 10.2, we conduct experiments using (1+ n)-EA, i.e., a specific case of (1+ λ)-EA with λ being the dimensionality n of the problem, on the Trap problem. We estimate the expected running time of (1+ n)-EA starting from a solution s with $|s|_0 = i$ for each i where $0 \leq i \leq n$. Following Theorem 10.2, we compare the estimated expected running time of (1+ n)-EA without noise, with additive noise and with multiplicative noise. For the mutation probability of (1+ n)-EA, the common setting $p = 1/n$ is used. For additive noise, $\delta_1 = -n$ and $\delta_2 = n - 1$, and for multiplicative noise, $\delta_1 = 0.1$ and $\delta_2 = 10$. The results for $n = 5, 6, 7$ are plotted in Figure 10.1. It can be observed that the curves by these two types of noise are always below the curve without noise, consistent with our theoretical result. Note that the three curves meet at the first point, because the initial solution with $|s|_0 = 0$ is the optimal solution.

As for Theorem 10.3, we conduct experiments using (1+1)-EA $^\neq$ on the Peak problem. One-bit noise is set with $p_n = 0.5$. The results for $n = 6, 7, 8$ are plotted in Figure 10.2. It can be observed that the curve with one-bit noise is always below the curve without noise when $|s|_0$ is large enough, consistent with our theoretical analysis.

We have shown that noise can make deceptive and flat problems easier for EAs. For deceptive problems, it is intuitively because the EA searches along the deceptive direction while noise can add some randomness to make the EA have some possibility to run along the right direction; for flat problems, the EA has no guided information for search while under some

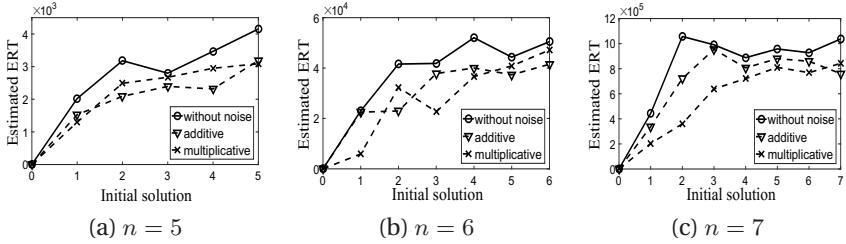


Figure 10.1. Estimated expected running time (ERT) comparison for $(1+n)$ -EA solving the Trap problem with/without noise.

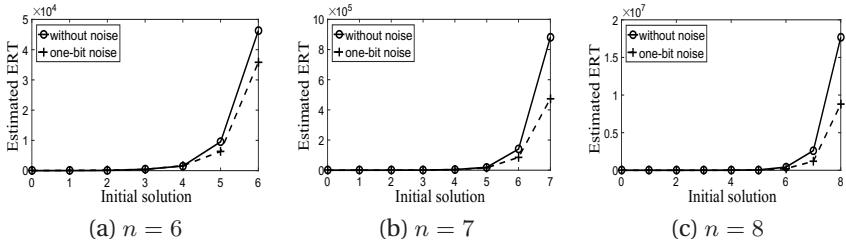


Figure 10.2. Estimated expected running time (ERT) comparison for $(1+1)$ -EA $^{\neq}$ solving the Peak problem with/without noise.

situations noise can make the EA have a larger probability to run along the right direction than the wrong direction.

Though the Trap and Peak problems are extremely deceptive and flat, respectively, in real applications optimization problems with some degree of deceptiveness and flatness are often encountered. We next test whether the findings on the extreme cases hold on other problems. We employ $(1+1)$ -EA with the mutation probability $1/n$ on the minimum spanning tree problem, where the solution representation and the fitness function have been introduced in Section 9.3. Each non-minimum spanning tree is local optimal in the Hamming space, and thus the minimum spanning tree problem is a multimodal problem with local deceptiveness.

We conduct experiments to compare $(1+1)$ -EA without noise and with one-bit noise where $p_n = 0.5$, on three types of graphs, i.e., sparse graphs, moderate graphs and dense graphs, which have been used in Section 9.4. Their number of edges is in the order of $\Theta(n)$, $\Theta(n\sqrt{n})$ and $\Theta(n^2)$, respectively. For each type of graph and each independent run, the graph is constructed by setting the weight of each edge to a value selected randomly from $[n]$. The results are plotted in Figure 10.3. It can be observed that the curves with one-bit noise can appear below the curves without noise, verifying our finding that noise can make problems easier for EAs.

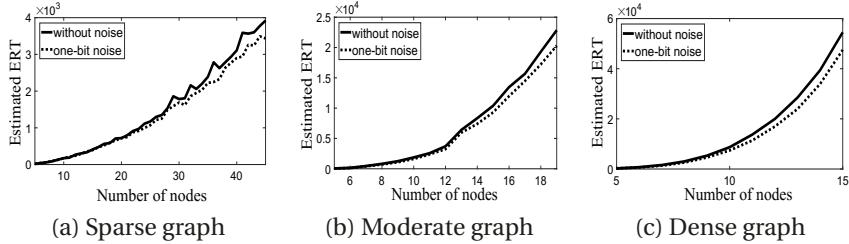


Figure 10.3. Estimated expected running time (ERT) comparison for (1+1)-EA solving the minimum spanning tree problem with/without noise.

10.5.2 Noise Harmful Cases

Next we verify our theoretical result that any noise will do harm to the One-Max problem for $(1+\lambda)$ -EA. The experimental setting is the same as that for $(1+\lambda)$ -EA on the Trap problem in Section 10.5.1. The results for $n = 10, 20, 30$ are plotted in Figure 10.4. It can be observed that the curve by any noise is always above the curve without noise, consistent with our theoretical result.

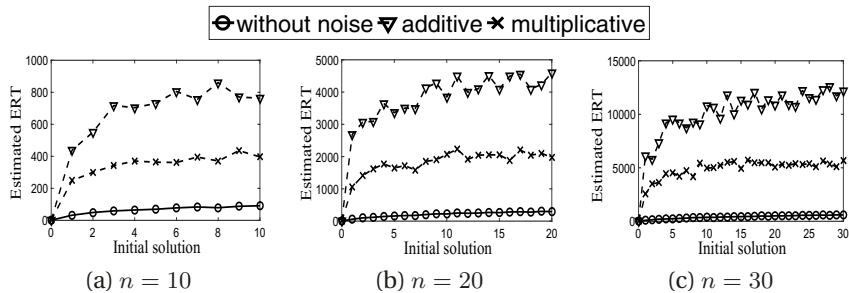


Figure 10.4. Estimated expected running time (ERT) comparison for $(1+n)$ -EA solving the OneMax problem with/without noise.

10.5.3 Discussion

We have derived that in the deceptive and flat cases noise can make the problem easier for EAs. The deceptiveness and flatness are two factors that can block the search of EAs, and the problems in these two cases are hard for EAs. We have also derived that noise can make the easy problem One-Max harder for EAs. Thus, we hypothesize that the negative influence of noise decreases as the problem hardness increases, and noise will bring a positive influence when the problem is quite hard. The influence of noise can be measured by the gap of estimated expected running time,

$$gap = (\mathbb{E}[\tau] - \mathbb{E}[\tau'])/\mathbb{E}[\tau'], \quad (10.46)$$

where $\mathbb{E}[\tau]/\mathbb{E}[\tau']$ denote the expected running time of the EA optimizing the problem with/without noise. Note that noise is harmful if the gap is positive, and helpful if the gap is negative.

To verify our hypothesis, we evaluate (1+1)-EA on the $\text{Jump}_{m,n}$ problem in Definition 10.8, as well as the minimum spanning tree problem.

Definition 10.8 (Jump_{m,n}). *The Jump_{m,n} Problem of size n with $1 \leq m \leq n$ is to find an n bits binary string which maximizes*

$$\text{Jump}_{m,n}(s) = \begin{cases} m + \sum_{i=1}^n s_i & \text{if } \sum_{i=1}^n s_i \leq n - m \text{ or } = n; \\ n - \sum_{i=1}^n s_i & \text{otherwise,} \end{cases} \quad (10.47)$$

where s_i denotes the i-th bit of $s \in \{0, 1\}^n$.

The $\text{Jump}_{m,n}$ problem has an adjustable hardness and can be configured as the OneMax problem when $m = 1$ and the Trap problem when $m = n$. The expected running time of (1+1)-EA on the $\text{Jump}_{m,n}$ problem is $\Theta(n^m + n \log n)$ [Droste et al., 2002], implying that the $\text{Jump}_{m,n}$ problem with larger value of m is harder. In the experiments, we set $n = 10$, and for noise, we use additive noise with $\delta_1 = -0.5n \wedge \delta_2 = 0.5n$, multiplicative noise with $\delta_1 = 1 \wedge \delta_2 = 2$, and one-bit noise with $p_n = 0.5$. The experimental results on gap values are plotted in Figure 10.5. We can have a clear observation that the gap values for larger m are lower, i.e., the negative influence by noise decreases as the problem hardness increases.

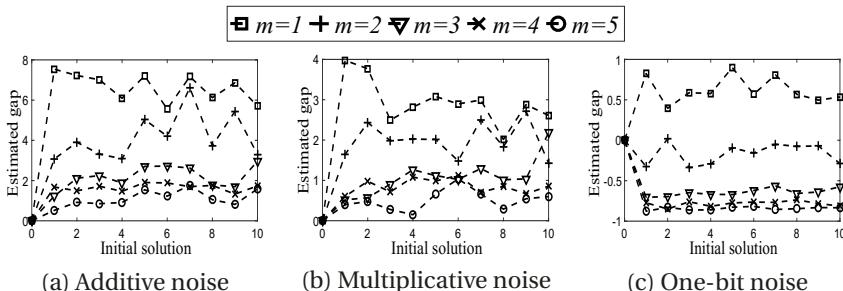


Figure 10.5. Estimated expected running time gap for (1+1)-EA solving the $\text{Jump}_{m,10}$ problem with/without noise.

The minimum spanning tree problem with sparse, moderate and dense graphs is evaluated. As shown in the last cell of the second row of Table 9.1, the expected running time of (1+1)-EA on this problem is $O(m^2(\log n +$

$\log w_{\max})$). With the assumption that this theoretical upper bound is tight, the hardness order of the three types of graphs is “sparse” < “moderate” < “dense”. The results are plotted in Figure 10.6. It can be observed that the height order of the curves is “sparse” > “moderate” > “dense”, consistent with our hypothesis.

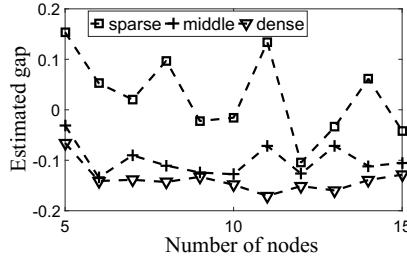


Figure 10.6. Estimated expected running time gap for (1+1)-EA solving the minimum spanning tree problem with/without one-bit noise.

Experiments on both the artificial $\text{Jump}_{m,n}$ problem and the combinatorial minimum spanning tree problem verify the same trend that the influence of noise can be related to the hardness of problem to the EA. When the problem is hard, noise can be helpful to the EA, and thus noise handling may be not necessary when the problem is hard.

10.6 Summary

In this chapter, we study the influence of noise on the expected running time of EAs, and prove that for the Trap and Peak problems, the presence of noise can decrease the expected running time of EAs, whereas for the One-Max problem, it increases the expected running time. These results suggest that the influence of noise is correlated with the problem hardness: When the problem is EA-hard with respect to a specific EA, noise can be helpful and does not need to be handled, whereas when the problem is EA-easy, noise can be harmful and needs to be tackled. These findings are verified by experiments on synthetic as well as combinatorial problems.

Next we show that the two common strategies, i.e., threshold selection and sampling, can bring robustness against noise. We prove that by using threshold selection with the threshold $\tau = 1$, (1+1)-EA can solve OneMax under one-bit noise in polynomial running time regardless of the probability p_n of noise. For $\tau \geq 2$, however, we prove that the ranges of p_n allowing a polynomial running time can be reduced largely. These results suggest that a careful selection of τ is important for the effectiveness of threshold selection. We also disclose a weakness of threshold selection: For solving One-Max under asymmetric one-bit noise with $p_n = 1$, (1+1)-EA using threshold

selection with any τ requires exponential running time. By turning the original hard threshold into a smooth one, we introduce the smooth threshold selection strategy, allowing a fractional threshold to be effective. We prove that with smooth threshold selection, (1+1)-EA is a polynomial time algorithm on the problem regardless of p_n .

For the sampling strategy, we show that it can speed up noisy evolutionary optimization exponentially. For (1+1)-EA solving the OneMax and LeadingOnes problems under one-bit or additive Gaussian noise, we prove that the running time is exponential when the level of noise is high, while sampling can reduce the running time to a polynomial. Particularly, for (1+1)-EA solving OneMax under one-bit noise with $p_n = 1$, the analysis also shows that a small change of the k value for sampling can lead to an exponential difference of the expected running time, disclosing that a careful selection of k is important for the effectiveness of sampling.

Note that the influence of selection has been studied for noiseless cases. For example, the expected running time of EAs using elitist selection, which keeps the best-so-far solution, or non-elitist selection, which can accept worse solutions, has been compared [Jägersküpper and Storch, 2007, Jansen and Wegener, 2007, Neumann et al., 2009, Oliveto et al., 2018], and the analysis reveals that non-elitist selection can be better sometimes.



Population

Most EAs maintain a population, i.e., a set of solutions, during the optimization process. This chapter studies the influence of the population size on the running time of EAs, and then whether using population, i.e., more than one solution, can bring robustness against noise.

On the aspect of population, previous running time analysis mostly focused on $(1+1)$ -EA, $(\mu+1)$ -EA which has only parent population, or $(1+\lambda)$ -EA which has only offspring population [Neumann and Witt, 2010, Auger and Doerr, 2011]. For example, Droste et al. [2002] proved that the expected running time of $(1+1)$ -EA on linear pseudo-Boolean functions is $\Theta(n \log n)$; for $(\mu+1)$ -EA solving several artificially designed functions, a large parent population size μ has been shown to be able to reduce the running time from exponential to polynomial [Jansen and Wegener, 2001, Storch, 2008, Witt, 2006, 2008]; for $(1+\lambda)$ -EA solving linear functions, the expected running time has been proved to be $O(n \log n + \lambda n)$ [Doerr and Künnemann, 2015], and a tighter bound up to lower order terms has been derived on the specific linear function OneMax [Gießen and Witt, 2015].

If both parent and offspring populations are involved, the running time analysis becomes more complicated, and only a few results have been reported on $(\lambda+\lambda)$ -EA, i.e., a specific version of $(\mu+\lambda)$ -EA in Algorithm 2.4 with $\mu = \lambda$, which maintains λ solutions and generates λ offspring solutions by mutation only in each iteration. He and Yao [2002] compared the expected running time of $(1+1)$ -EA and $(\lambda+\lambda)$ -EA on two specific artificial problems, and proved that the introduction of population can reduce the running time exponentially. On the other hand, Chen et al. [2012] found that a large population size can be harmful for $(\lambda+\lambda)$ -EA solving the TrapZeros problem. Chen et al. [2009a] also proved that the expected running time of $(\lambda+\lambda)$ -EA on the OneMax and LeadingOnes problems is $O(\lambda n \log \log n + n \log n)$ and $O(\lambda n \log n + n^2)$, respectively. Later, a low selection pressure has been shown to be better for $(\lambda+\lambda)$ -EA solving a wide gap problem [Chen et al., 2010a], and a proper mutation-selection balance

has been proved to be necessary for the effectiveness of $(\lambda+\lambda)$ -EA solving the SelPres problem [Lehre and Yao, 2012].

The studies mentioned above on $(\lambda+\lambda)$ -EA usually focus on specific functions, though EAs are general-purpose optimization algorithms that can be applied to any optimization problem where solutions can be represented and evaluated. Thus, it is desired to analyze EAs over large problem classes. Meanwhile, previous studies on population-based EAs generally derived only upper bounds. Although upper bounds are appealing for revealing the ability of an algorithm, lower bounds which reveal the limitation is also important for a complete understanding of the algorithm.

In this chapter, we present the running time analysis of $(\mu+\lambda)$ -EA for solving the UBoolean function class in Definition 2.2 [Qian et al., 2016b]. By applying the switch analysis approach in Theorem 4.1, we prove that the expected running time is $\Omega(n \log n + \mu + \lambda n \log \log n / \log n)$. Particularly, when applying this lower bound to the two specific problems, i.e., OneMax and LeadingOnes, we get a complete understanding of the influence of the offspring population size λ . It has been known that $(\mu+\lambda)$ -EA is not asymptotically faster than $(1+1)$ -EA on these two problems [Lehre and Witt, 2012, Sudholt, 2013], but it remains unclear about what the range of λ is for $(\mu+\lambda)$ -EA to be asymptotically slower than $(1+1)$ -EA. Note that the expected running time of $(1+1)$ -EA on OneMax and LeadingOnes is $\Theta(n \log n)$ and $\Theta(n^2)$, respectively [Droste et al., 2002]. Thus, $(\mu+\lambda)$ -EA is asymptotically slower than $(1+1)$ -EA when $\lambda = \omega((\log n)^2 / \log \log n)$ on OneMax and $\lambda = \omega(n \log n / \log \log n)$ on LeadingOnes. For the parent population size μ , we get the ranges $\omega(n \log n)$ and $\omega(n^2)$ for $(\mu+\lambda)$ -EA to be asymptotically slower on OneMax and LeadingOnes, respectively. Our results disclose that the increase of population size, though usually desired in practice, bears the risk of increasing the lower bound of the running time and thus should be carefully considered.

Next we present the robustness analysis of population against noise [Qian et al., 2018a], when EAs are used for noisy optimization. In Section 10.4, we have shown that sampling, which evaluates the noisy fitness of a solution multiple times independently and uses the average for estimation, is an effective strategy to tackle noise. By two illustrative examples, we show that using parent or offspring population can be better than using sampling.

The rest of this chapter is organized as follows. Section 11.1 presents the running time analysis of $(\mu+\lambda)$ -EA on UBoolean to study the influence of population. Section 11.2 presents the robustness analysis of population against noise. Section 11.3 concludes this chapter.

11.1 Influence of Population

In this section, we present a lower bound of the expected running time for $(\mu+\lambda)$ -EA solving UBoolean in Theorem 11.1. Our proof is accomplished by

using switch analysis in Theorem 4.1. The target EA process to be analyzed is $(\mu+\lambda)$ -EA in Algorithm 2.4 running on any function in UBoolean. Note that for any function in UBoolean, we assume without loss of generality that the optimal solution is 1^n . The constructed reference process for comparison is the RLS $^{\neq}$ algorithm running on the LeadingOnes problem. As introduced in Section 2.1, RLS $^{\neq}$, which is a modification of the RLS algorithm, maintains only one solution s . In each iteration, a new solution s' is generated by flipping a randomly chosen bit of s , and s' is accepted only if $f(s') > f(s)$. In other words, RLS $^{\neq}$ searches locally and accepts only a better offspring solution. For the CFHT $\mathbb{E}[\tau' \mid \xi'_t = y]$ of the reference chain $\xi' \sim \mathcal{Y}$ modeling RLS $^{\neq}$ on LeadingOnes, Lemma 5.8 shows that $\mathbb{E}[\tau' \mid \xi'_t = y]$ with $|y|_0 = j$ can be denoted by $\mathbb{E}_{rls}(j)$, which is equal to nj .

Before proving Theorem 11.1, we first give some lemmas that will be used. The proofs of Lemmas 11.1 and 11.2 are provided in Appendix A.6.

Lemma 11.1. *For $m \geq i \geq 0$, it holds that $\sum_{k=0}^i \binom{m}{k} (1/n)^k (1 - 1/n)^{m-k}$ decreases with m .*

Lemma 11.2. *For $\lambda \leq n^c$ where c is a positive constant, it holds that*

$$\sum_{i=0}^{n-1} \left(\sum_{k=0}^i \binom{n}{k} \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \right)^\lambda \geq n - \left\lceil \frac{e(c+1) \log n}{\log \log n} \right\rceil. \quad (11.1)$$

Lemma 11.3. [Flum and Grohe, 2006] *Let $H(\epsilon) = -\epsilon \log \epsilon - (1-\epsilon) \log(1-\epsilon)$, where the base of the logarithm is 2. It holds that*

$$\forall n \geq 1, 0 < \epsilon < \frac{1}{2} : \sum_{k=0}^{\lfloor \epsilon n \rfloor} \binom{n}{k} \leq 2^{H(\epsilon)n}. \quad (11.2)$$

Theorem 11.1. *The expected running time of $(\mu+\lambda)$ -EA solving UBoolean is $\Omega(n \log n + \mu + \lambda n \log \log n / \log n)$, when μ and λ are upper bounded by a polynomial in n .*

Proof. Let $\xi \in \mathcal{X}$ model the concerned EA process, i.e., $(\mu+\lambda)$ -EA running on any function in UBoolean. We use RLS $^{\neq}$ running on the LeadingOnes problem as the reference process modeled by $\xi' \in \mathcal{Y}$. We have $\mathcal{Y} = \{0,1\}^n$, $\mathcal{X} = \{\{s_1, s_2, \dots, s_\mu\} \mid s_i \in \{0,1\}^n\}$, $\mathcal{Y}^* = \{1^n\}$ and $\mathcal{X}^* = \{x \in \mathcal{X} \mid \min_{s \in x} |s|_0 = 0\}$. We construct a mapping $\phi : \mathcal{X} \rightarrow \mathcal{Y}$ as $\forall x \in \mathcal{X} : \phi(x) = \arg \min_{s \in x} |s|_0$. The mapping is an optimal-aligned mapping because $\phi(x) \in \mathcal{Y}^*$ iff $x \in \mathcal{X}^*$.

We examine the condition, i.e., Eq. (4.1), of switch analysis. $\forall x \notin \mathcal{X}^*$, suppose $\min\{|s|_0 \mid s \in x\} = j > 0$, implying $|\phi(x)|_0 = j$. By Lemmas 2.1 and 5.8, we have

$$\sum_{y \in \mathcal{Y}} P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] = \mathbb{E}_{rls}(j) - 1 = nj - 1. \quad (11.3)$$

For the reproduction of $(\mu+\lambda)$ -EA, i.e., the chain $\xi \in \mathcal{X}$, on the population x , assume that the λ solutions selected from x for reproduction have the number of 0-bits $j_1, j_2, \dots, j_\lambda$, respectively, where $j \leq j_1 \leq j_2 \leq \dots \leq j_\lambda \leq n$. If there are at most i ($0 \leq i \leq j_1$) number of 0-bits mutated to 1-bits for each selected solution and there exists at least one selected solution which flips exactly i number of 0-bits, occurring with probability $\prod_{p=1}^{\lambda} (\sum_{k=0}^i \binom{j_p}{k}) (1/n)^k (1 - 1/n)^{j_p - k} - \prod_{p=1}^{\lambda} (\sum_{k=0}^{i-1} \binom{j_p}{k}) (1/n)^k (1 - 1/n)^{j_p - k}$, denoted by $p(i)$, the next population x' satisfies $|\phi(x')|_0 \geq j_1 - i$. Furthermore, $\mathbb{E}_{rls}(i) = ni$ increases with i . Thus, we have

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ & \geq \sum_{i=0}^{j_1} p(i) \cdot \mathbb{E}_{rls}(j_1 - i) \geq \sum_{i=0}^j p(i) \cdot \mathbb{E}_{rls}(j - i) \\ & = n \sum_{i=0}^{j-1} \left(\prod_{p=1}^{\lambda} \left(\sum_{k=0}^i \binom{j_p}{k} \left(\frac{1}{n} \right)^k \left(1 - \frac{1}{n} \right)^{j_p - k} \right) \right). \end{aligned} \quad (11.4)$$

By comparing Eqs. (11.3) and (11.4), we have, $\forall x \notin \mathcal{X}^*$,

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ & - \sum_{y \in \mathcal{Y}} P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] \\ & \geq n \left(\sum_{i=0}^{j-1} \left(\prod_{p=1}^{\lambda} \left(\sum_{k=0}^i \binom{j_p}{k} \left(\frac{1}{n} \right)^k \left(1 - \frac{1}{n} \right)^{j_p - k} \right) \right) - j \right) + 1 \\ & \geq n \left(\sum_{i=0}^{j-1} \left(\sum_{k=0}^i \binom{n}{k} \left(\frac{1}{n} \right)^k \left(1 - \frac{1}{n} \right)^{n-k} \right)^{\lambda} - j \right) + 1 \end{aligned} \quad (11.5)$$

$$\geq n \left(\sum_{i=0}^{n-1} \left(\sum_{k=0}^i \binom{n}{k} \left(\frac{1}{n} \right)^k \left(1 - \frac{1}{n} \right)^{n-k} \right)^{\lambda} - n \right) + 1, \quad (11.6)$$

where Eq. (11.5) holds by Lemma 11.1, i.e., $\sum_{k=0}^i \binom{m}{k} (1/n)^k (1 - 1/n)^{m-k}$ reaches the minimum when $m = n$, and Eq. (11.6) holds by $(\sum_{k=0}^i \binom{n}{k} (1/n)^k (1 - 1/n)^{n-k})^{\lambda} \leq 1$. When $x \in \mathcal{X}^*$, Eqs. (11.3) and (11.4) equal 0, because both chains are absorbing and the mapping ϕ is optimal-aligned. Thus, Eq. (4.1) in Theorem 4.1 holds with $\rho_t = (n(\sum_{i=0}^{n-1} (\sum_{k=0}^i \binom{n}{k} (1/n)^k (1 - 1/n)^{n-k})^{\lambda} - n) + 1)(1 - \pi_t(\mathcal{X}^*))$. By Theorem 4.1, we have

$$\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \geq \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^{\phi}] \quad (11.7)$$

$$+ \left(n \left(\sum_{i=0}^{n-1} \left(\sum_{k=0}^i \binom{n}{k} \left(\frac{1}{n} \right)^k \left(1 - \frac{1}{n} \right)^{n-k} \right)^\lambda - n \right) + 1 \right) \sum_{t=0}^{+\infty} (1 - \pi_t(\mathcal{X}^*)).$$

Considering $\sum_{t=0}^{+\infty} (1 - \pi_t(\mathcal{X}^*)) = \mathbb{E}[\tau \mid \xi_0 \sim \pi_0]$, we have

$$\begin{aligned} \mathbb{E}[\tau \mid \xi_0 \sim \pi_0] &\geq \frac{\mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi]}{n \left(n - \sum_{i=0}^{n-1} \left(\sum_{k=0}^i \binom{n}{k} \left(\frac{1}{n} \right)^k \left(1 - \frac{1}{n} \right)^{n-k} \right)^\lambda \right)} \\ &\geq \frac{\mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi]}{n \left\lceil \frac{e(c+1) \log n}{\log \log n} \right\rceil}, \end{aligned} \quad (11.8)$$

where Eq. (11.8) holds by Lemma 11.2, as $\lambda \leq n^c$ for some constant c .

Next we examine $\mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi]$. Because each of the μ solutions in the initial population is selected uniformly randomly from $\{0, 1\}^n$, we have

$$\begin{aligned} \forall 0 \leq j \leq n : \pi_0^\phi(\{y \in \mathcal{Y} \mid |y|_0 = j\}) &= \pi_0(\{x \in \mathcal{X} \mid \min_{y \in \mathcal{X}} |y|_0 = j\}) \quad (11.9) \\ &= \frac{\left(\sum_{k=j}^n \binom{n}{k} \right)^\mu - \left(\sum_{k=j+1}^n \binom{n}{k} \right)^\mu}{2^{n\mu}}, \end{aligned}$$

where $\sum_{k=j}^n \binom{n}{k}$ is the number of solutions with no less than j number of 0-bits. Then, we have

$$\begin{aligned} \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] &= \sum_{j=0}^n \pi_0^\phi(\{y \in \mathcal{Y} \mid |y|_0 = j\}) \mathbb{E}_{rls}(j) \\ &= \frac{1}{2^{n\mu}} \sum_{j=1}^n \left(\left(\sum_{k=j}^n \binom{n}{k} \right)^\mu - \left(\sum_{k=j+1}^n \binom{n}{k} \right)^\mu \right) nj \\ &= \frac{n}{2^{n\mu}} \sum_{j=1}^n \left(\sum_{k=j}^n \binom{n}{k} \right)^\mu > n \sum_{j=1}^{\lfloor \frac{n}{4} \rfloor + 1} \left(\sum_{k=j}^{\lfloor \frac{n}{4} \rfloor + 1} \frac{\binom{n}{k}}{2^n} \right)^\mu \\ &> \frac{n^2}{4} \left(\sum_{k=\lfloor \frac{n}{4} \rfloor + 1}^n \frac{\binom{n}{k}}{2^n} \right)^\mu = \frac{n^2}{4} \left(1 - \sum_{k=0}^{\lfloor \frac{n}{4} \rfloor} \frac{\binom{n}{k}}{2^n} \right)^\mu \\ &\geq \frac{n^2}{4} \left(1 - 2^{H(1/4)n-n} \right)^\mu \end{aligned} \quad (11.10)$$

$$\geq \frac{n^2}{4} e^{-\frac{\mu}{2^{(1-H(1/4))n-1}}} \quad (11.11)$$

$$> \frac{n^2}{4} e^{-\frac{\mu}{1.13^n-1}}, \quad (11.12)$$

where Eq. (11.10) holds by Lemma 11.3, Eq. (11.11) holds by $\forall 0 < x < 1 : (1-x)^y \geq e^{-xy/(1-x)}$, and Eq. (11.12) holds by $2^{1-H(1/4)} > 1.13$.

Applying the above lower bound of $\mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi]$ to Eq. (11.8) and considering that μ is upper bounded by a polynomial in n , we have

$$\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \geq \frac{n}{4 \left\lceil \frac{e(c+1) \log n}{\log \log n} \right\rceil} e^{-\frac{\mu}{1.13^{n-1}}} = \Omega \left(\frac{n \log \log n}{\log n} \right). \quad (11.13)$$

Considering the μ fitness evaluations for the initial population and the λ fitness evaluations in each generation, the expected running time of $(\mu+\lambda)$ -EA on UBoolean is lower bounded by $\Omega(\mu + \lambda n \log \log n / \log n)$. Because $(\mu+\lambda)$ -EA belongs to mutation-based EAs, the general lower bound $\Omega(n \log n)$ [Sudholt, 2013] can be directly applied. Thus, the theorem holds. \square

11.2 Robustness of Population against Noise

Gießen and Kötzing [2016] have shown that using population can increase robustness against noise. For example, for the OneMax problem under one-bit noise with $p_n = \omega(\log n/n)$, (1+1)-EA requires super-polynomial time to find the optimal solution as shown in Theorem 10.5, whereas using a parent population size $\mu \geq 12 \log(15n)/p_n$ or an offspring population size $\lambda \geq \max\{12/p_n, 24\}n \log n$ can reduce the running time to a polynomial [Gießen and Kötzing, 2016]. In Section 10.4, we also show the robustness of sampling against noise. For example, for (1+1)-EA solving OneMax under one-bit noise with $p_n = 1$, using a sample size $k = 3$ can reduce the running time to a polynomial. In this section, we prove that the increase of robustness against noise by using population can be more effective than using sampling.

11.2.1 Parent Population

First, we show that compared with using sampling, using parent population can be more robust against noise. Particularly, we compare (1+1)-EA using sampling with $(\mu+1)$ -EA in Algorithm 2.2 for solving OneMax under symmetric noise. The OneMax problem in Definition 2.3 is to maximize the number of 1-bits of a solution, and the optimal solution is 1^n . As presented in Definition 11.1, symmetric noise returns a false fitness $(2n - f(s))$ with probability 1/2. Under this noise model, the distribution of $f^n(s)$ for any s is symmetric about n .

Definition 11.1 (Symmetric Noise). Let $f^n(s)$ and $f(s)$ denote the noisy and true fitness of a solution s , respectively, then

$$f^n(s) = \begin{cases} f(s) & \text{with probability } 1/2; \\ 2n - f(s) & \text{with probability } 1/2. \end{cases} \quad (11.14)$$

We prove in Theorem 11.2 that the expected running time of (1+1)-EA using sampling with any sample size k is exponential. From the proof, we can find the reason why using sampling fails. Under symmetric noise, the distribution of $f^n(s)$ for any s is symmetric about n . Thus, for any two solutions s and s' , the distribution of $(f^n(s) - f^n(s'))$ is symmetric about 0. By using sampling, the distribution of $(\hat{f}(s) - \hat{f}(s'))$ is still symmetric about 0, implying that the offspring solution will always be accepted with probability at least 1/2 in each iteration of (1+1)-EA. Such a behavior is analogous to random walk, and thus the optimization is inefficient.

Theorem 11.2. *For (1+1)-EA solving OneMax under symmetric noise, if using sampling, the expected running time is exponential.*

Proof. We use Theorem 2.5 for the proof. Let $x_t = |s|_0$ be the number of 0-bits of the solution s after t iterations of (1+1)-EA. We consider the interval $[0, n/10]$, i.e., the parameters $a = 0$ and $b = n/10$ in Theorem 2.5.

We analyze the drift $\mathbb{E}[x_t - x_{t+1} \mid x_t = i]$ for $1 \leq i < n/10$. Let $P_{\text{mut}}(s, s')$ denote the probability of generating s' by mutating s . We divide the drift into two parts: positive E^+ and negative E^- ; that is,

$$\mathbb{E}[x_t - x_{t+1} \mid x_t = i] = E^+ - E^-, \quad (11.15)$$

$$\text{where } E^+ = \sum_{s': |s'|_0 < i} P_{\text{mut}}(s, s') \cdot P(\hat{f}(s') \geq \hat{f}(s)) \cdot (i - |s'|_0), \quad (11.16)$$

$$E^- = \sum_{s': |s'|_0 > i} P_{\text{mut}}(s, s') \cdot P(\hat{f}(s') \geq \hat{f}(s)) \cdot (|s'|_0 - i). \quad (11.17)$$

For E^+ , consider that the number of 0-bits is decreased. For mutation on s , where $|s|_0 = i$, let x and y denote the number of flipped 0-bits and 1-bits, respectively. Then, $x \sim B(i, 1/n)$ and $y \sim B(n-i, 1/n)$, where $B(\cdot, \cdot)$ is the binomial distribution. To estimate an upper bound of E^+ , assume that the offspring solution s' with $|s'|_0 < i$ is always accepted. Thus, we have

$$\begin{aligned} E^+ &\leq \sum_{s': |s'|_0 < i} P_{\text{mut}}(s, s') (i - |s'|_0) \\ &= \sum_{k=1}^i k \cdot P(x - y = k) = \sum_{k=1}^i k \cdot \sum_{j=k}^i P(x = j) \cdot P(y = j - k) \\ &= \sum_{j=1}^i \sum_{k=1}^j k \cdot P(x = j) \cdot P(y = j - k) \leq \sum_{j=1}^i j \cdot P(x = j) = \frac{i}{n}. \end{aligned} \quad (11.18)$$

For E^- , consider that the number of 0-bits is increased. We analyze the $(n-i)$ cases where only one 1-bit is flipped, i.e., $|s'|_0 = i+1$, occurring with probability $(1/n)(1-1/n)^{n-1} \geq 1/(en)$. Let $y = f^n(s) - f^n(s')$. By the definition of symmetric noise, the value of y can be $-2i-1, -1, 1$ and $2i+1$, each

with probability $1/4$. The distribution of y is symmetric about 0, i.e., y has the same distribution as $-y$. Because $(\hat{f}(s) - \hat{f}(s'))$ is the average of k independent random variables, with the same distribution as y , the distribution of $(\hat{f}(s) - \hat{f}(s'))$ is also symmetric about 0, and thus $P(\hat{f}(s') \geq \hat{f}(s)) \geq 1/2$. Then, we have

$$E^- \geq \frac{n-i}{en} \cdot \frac{1}{2} \cdot (i+1-i) = \frac{n-i}{2en}. \quad (11.19)$$

By subtracting E^- from E^+ , we have

$$\mathbb{E}[x_t - x_{t+1} \mid x_t = i] \leq \frac{i}{n} - \frac{n-i}{2en} \leq -0.05, \quad (11.20)$$

where the last inequality holds by $i < n/10$. Thus, condition (1) of Theorem 2.5 holds with $\epsilon = 0.05$.

As the analysis of Eq. (A.98) in the proof of Theorem 10.9, condition (2) of Theorem 2.5 holds with $\delta = 1$ and $r(l) = 2$. Note that $l = b - a = n/10$. Thus, by Theorem 2.5, the expected running time is exponential. \square

We prove in Theorem 11.3 that $(\mu+1)$ -EA with $\mu = 3 \log n$ can find the optimal solution in $O(n \log^3 n)$ expected running time. The reason for the effectiveness of using parent population may be that the true best solution will be discarded only if it appears worse than all other solutions in the population, the probability of which can be very small by using a logarithmic parent population size. Note that this finding is consistent with that in [Gießen and Kötzing, 2016].

Theorem 11.3. *For $(\mu+1)$ -EA solving OneMax under symmetric noise, if $\mu = 3 \log n$, the expected running time is $O(n \log^3 n)$.*

Proof. We use Theorem 2.4 for the proof. Note that the state x of the corresponding Markov chain is currently a population, i.e., a set of μ solutions. We first construct a distance function V as $\forall x : V(x) = \min_{s \in x} |s|_0$, i.e., the minimum number of 0-bits of solutions in x . $V(x) = 0$ iff $x \in \mathcal{X}^*$, i.e., x contains the optimal solution 1^n .

We examine $\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x]$ $\forall x$ with $V(x) > 0$, i.e., $x \notin \mathcal{X}^*$. Assume that currently $V(x) = i$, where $1 \leq i \leq n$. The drift is also divided into two parts:

$$\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] = E^+ - E^-, \quad (11.21)$$

$$\text{where } E^+ = \sum_{x' : V(x') < i} P(\xi_{t+1} = x' \mid \xi_t = x) \cdot (i - V(x')), \quad (11.22)$$

$$E^- = \sum_{x' : V(x') > i} P(\xi_{t+1} = x' \mid \xi_t = x) \cdot (V(x') - i). \quad (11.23)$$

To examine E^+ , consider that the best solution in x is improved. Let $s^* = \arg \min_{s \in x} |s|_0$, implying $|s^*|_0 = i$. In one iteration of $(\mu+1)$ -EA, a solution s'

with $|s'|_0 = i - 1$ can be generated by selecting s^* and flipping only one 0-bit in mutation, occurring with probability $(1/\mu) \cdot (i/n)(1 - 1/n)^{n-1} \geq i/(e\mu n)$. If s' is not added into x , it must hold that $\forall s \in x : f^n(s') < f^n(s)$, occurring with probability $1/2^\mu$ as $f^n(s') < f^n(s)$ iff $f^n(s) = 2n - f(s)$. Thus, the probability that s' is added into x which implies $V(x') = i - 1$, is $1 - 1/2^\mu$. We then have

$$E^+ \geq \frac{i}{e\mu n} \cdot \left(1 - \frac{1}{2^\mu}\right) \cdot (i - (i - 1)) = \frac{i}{e\mu n} \left(1 - \frac{1}{2^\mu}\right). \quad (11.24)$$

For E^- , if there are at least two solutions s, s' in x such that $|s|_0 = |s'|_0 = i$, it holds that $E^- = 0$. Otherwise, $V(x') > V(x) = i$ implies that for the unique best solution s^* in x and $\forall s \in x \setminus \{s^*\}$, $f^n(s^*) \leq f^n(s)$, occurring with probability $1/2^{\mu-1}$ as $f^n(s^*) \leq f^n(s)$ iff $f^n(s) = 2n - f(s)$. Thus, $P(V(x') > i) \leq 1/2^{\mu-1}$. Furthermore, $V(x')$ can increase by at most $n - i$. Thus, $E^- \leq (n - i)/2^{\mu-1}$. By subtracting E^- from E^+ , we have

$$\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t] \geq \frac{i}{e\mu n} - \frac{i}{e\mu n 2^\mu} - \frac{n - i}{2^{\mu-1}} \geq \frac{i}{10n \log n}, \quad (11.25)$$

where the last inequality holds with sufficiently large n . By Theorem 2.4,

$$\mathbb{E}[\tau \mid \xi_0] \leq 10n \log n(1 + \log n) = O(n \log^2 n), \quad (11.26)$$

implying that the expected running time is $O(n \log^3 n)$, because the algorithm requires to evaluate the offspring solution and re-evaluate the $\mu = 3 \log n$ parent solutions in each iteration. \square

In the following, we show that the parent population size $\mu = 3 \log n$ is almost tight for making $(\mu+1)$ -EA efficient. Particularly, we prove that $\mu = O(1)$ is insufficient. Note that the proof is finished by applying the original negative drift analysis approach in Theorem 2.7 rather than the simplified versions in Theorems 2.5 and 2.6. To apply the simplified negative drift analysis approaches, we have to show that the probability of jumping towards and away from the target is exponentially decaying. However, the probability of jumping away from the target is at least a constant in this concerned case. To jump away from the target, it is sufficient that one non-best solution in the current population is cloned by mutation and then the best solution is deleted in updating the population. The former event occurs with probability $((\mu - 1)/\mu) \cdot (1 - 1/n)^n = \Theta(1)$, and the latter occurs with probability $1/2^\mu$, which is $\Theta(1)$ for $\mu = O(1)$. The original negative drift analysis approach is stronger than the simplified ones, and can be applied here to prove the exponential running time.

Theorem 11.4. *For $(\mu+1)$ -EA solving OneMax under symmetric noise, if $\mu = O(1)$, the expected running time is exponential.*

Proof. We use Theorem 2.7 for the proof. Let $x_t = y_t - h(z_t)$, where $y_t = \min_{s \in P} |s|_0$ denotes the minimum number of 0-bits of solutions in the population P after t iterations of $(\mu+1)$ -EA, $z_t = |\{s \in P \mid |s|_0 = y_t\}|$ denotes the number of solutions in P that have the minimum 0-bits y_t , and $\forall i \in [\mu] : h(i) = (d^{\mu-1} - d^{\mu-i})/(d^\mu - 1)$ with $d = 2^{\mu+4}$. Note that $0 = h(1) < h(2) < \dots < h(\mu) < 1$, and $x_t \leq 0$ iff $y_t = 0$, i.e., P contains at least one copy of the optimal solution 1^n . We set $l = n$, $\lambda(l) = 1$ and consider the interval $[0, cn - 1]$, where $c = 1/(3d^\mu)$, i.e., the parameters $a(l) = 0$ and $b(l) = cn - 1$ in Theorem 2.7.

We examine Eq. (2.33), which is equivalent to the following equation:

$$\sum_{r \neq x_t} P(x_{t+1} = r \mid a(l) < x_t < b(l)) \cdot (e^{x_t - r} - 1) \leq -\frac{1}{p(l)}. \quad (11.27)$$

We divide the term in the left side of Eq. (11.27) into two parts: $r < x_t$, i.e., $x_{t+1} < x_t$, and $r > x_t$, i.e., $x_{t+1} > x_t$.

First, consider $x_{t+1} < x_t$. Because $x_{t+1} = y_{t+1} - h(z_{t+1})$, $x_t = y_t - h(z_t)$ and $0 \leq h(z_{t+1}), h(z_t) < 1$, we have $x_{t+1} < x_t$ iff $y_{t+1} - y_t < 0$ or $y_{t+1} = y_t \wedge h(z_{t+1}) > h(z_t)$. We consider these two cases separately.

(1) $y_{t+1} - y_t = -j \leq -1$. It implies that a new solution s' with $|s'|_0 = y_t - j$ is generated in the $(t + 1)$ -th iteration of the algorithm. Suppose that s' is generated from some solution s , which must satisfy $|s|_0 \geq y_t$, selected from P , we have

$$\sum_{s' : |s'|_0 = y_t - j} P_{\text{mut}}(s, s') \leq \sum_{s' : |s'|_0 = y_t - j} P_{\text{mut}}(s^{y_t}, s') \quad (11.28)$$

$$\leq \binom{y_t}{j} \cdot \frac{1}{n^j} \quad (11.29)$$

$$\leq \left(\frac{y_t}{n}\right)^j \leq c^j, \quad (11.30)$$

where Eq. (11.28) holds by letting s^j denote any solution with j 0-bits, Eq. (11.29) holds as it requires to flip at least j 0-bits, and Eq. (11.30) holds by $y_t = x_t + h(z_t) < b(l) + 1 = cn$. Furthermore, according to $h(z_{t+1}) = h(1) = 0$, we have

$$x_t - x_{t+1} = y_t - h(z_t) - y_{t+1} + h(z_{t+1}) = j - h(z_t) \leq j. \quad (11.31)$$

(2) $y_{t+1} = y_t \wedge h(z_{t+1}) > h(z_t)$. It implies that $z_t < \mu$ and a new solution s' with $|s'|_0 = y_t$ is generated. Suppose that in the $(t + 1)$ -th iteration, the solution selected from P for mutation is s . If $|s|_0 > y_t$, we have

$$\begin{aligned} \sum_{s' : |s'|_0 = y_t} P_{\text{mut}}(s, s') &\leq \sum_{s' : |s'|_0 = y_t} P_{\text{mut}}(s^{y_t+1}, s') \\ &\leq \binom{y_t + 1}{1} \cdot \frac{1}{n} = \frac{y_t + 1}{n}. \end{aligned} \quad (11.32)$$

If $|s|_0 = y_t$, we have

$$\begin{aligned} \sum_{s':|s'|_0=y_t} P_{\text{mut}}(s, s') &\leq \left(1 - \frac{1}{n}\right)^n + \sum_{j=1}^{y_t} \binom{y_t}{j} \cdot \frac{1}{n^j} \\ &\leq \frac{1}{e} + \sum_{j=1}^{y_t} \left(\frac{y_t}{n}\right)^j \leq \frac{1}{e} + \frac{y_t/n}{1 - y_t/n} = \frac{1}{e} + \frac{1}{n/y_t - 1}. \end{aligned} \quad (11.33)$$

As $y_t = x_t + h(z_t) < b(l) + 1 = cn$ and $c = 1/(3d^\mu) = 1/(3 \cdot 2^{\mu(\mu+4)})$, we have

$$\sum_{s':|s'|_0=y_t} P_{\text{mut}}(s, s') \leq \frac{1}{2}. \quad (11.34)$$

Furthermore, it must hold that $z_{t+1} = z_t + 1$, and thus,

$$x_t - x_{t+1} = h(z_{t+1}) - h(z_t) = h(z_t + 1) - h(z_t). \quad (11.35)$$

By combining the above two cases, we have

$$\begin{aligned} &\sum_{r < x_t} P(x_{t+1} = r \mid a(l) < x_t < b(l)) \cdot (e^{x_t - r} - 1) \\ &\leq \sum_{j=1}^{y_t} c^j \cdot (e^j - 1) + \begin{cases} \frac{1}{2} \cdot (e^{h(z_t+1)-h(z_t)} - 1), & z_t < \mu \\ 0, & z_t = \mu \end{cases} \\ &\leq \sum_{j=1}^{y_t} (ce)^j + \begin{cases} h(z_t + 1) - h(z_t), & z_t < \mu \\ 0, & z_t = \mu \end{cases} \end{aligned} \quad (11.36)$$

$$\leq \frac{ce}{1-ce} + \begin{cases} h(z_t + 1) - h(z_t), & z_t < \mu \\ 0, & z_t = \mu \end{cases}. \quad (11.37)$$

Eq. (11.36) holds by $h(z_t + 1) - h(z_t) \in (0, 1)$ and $e^x \leq 1 + 2x$ for $x \in (0, 1)$.

Next, consider $x_{t+1} > x_t$, which holds iff in the $(t+1)$ -th iteration, the newly generated solution s' satisfies $|s'|_0 > y_t$ and one solution s^* in P with $|s^*|_0 = y_t$ is deleted. We first analyze the probability of generating a new solution s' with $|s'|_0 > y_t$. Suppose that the solution selected from P for mutation is s . If $|s|_0 > y_t$, it is sufficient that all bits of s are not flipped, and thus $\sum_{s':|s'|_0>y_t} P_{\text{mut}}(s, s') \geq (1 - 1/n)^n \geq (n-1)/(en)$. If $|s|_0 = y_t$, it is sufficient that only one 1-bit of s is flipped, and thus $\sum_{s':|s'|_0>y_t} P_{\text{mut}}(s, s') \geq (1 - 1/n)^{n-1}(n - y_t)/n \geq (n - y_t)/(en)$. Considering that $y_t = x_t + h(z_t) < b(l) + 1 = cn$ and $c = 1/(3 \cdot 2^{\mu(\mu+4)}) = \Theta(1)$ for $\mu = O(1)$, we have

$$\sum_{s':|s'|_0>y_t} P_{\text{mut}}(s, s') \geq \frac{1-c}{e}. \quad (11.38)$$

The probability of deleting one solution s^* in P with $|s^*|_0 = y_t$ is at least $1/2^\mu$, because it is sufficient to consider the situation that the fitness evaluation of all solutions in $P \cup \{s'\}$ with more than y_t 0-bits is affected by noise. We then analyze $x_t - x_{t+1}$. If $z_t = 1$, we have $y_{t+1} \geq y_t + 1$, and thus,

$$x_t - x_{t+1} = y_t - y_{t+1} + h(z_{t+1}) - h(z_t) \leq h(\mu) - 1. \quad (11.39)$$

If $z_t \geq 2$, we have $y_{t+1} = y_t$ and $z_{t+1} = z_t - 1$, and thus,

$$x_t - x_{t+1} = h(z_{t+1}) - h(z_t) = h(z_t - 1) - h(z_t). \quad (11.40)$$

Note that for $x_{t+1} > x_t$, $e^{x_t - x_{t+1}} - 1 < 0$. Thus, we have

$$\begin{aligned} & \sum_{r>x_t} P(x_{t+1} = r \mid a(l) < x_t < b(l)) \cdot (e^{x_t - r} - 1) \\ & \leq \frac{1}{2^\mu} \cdot \frac{1-c}{e} \cdot \begin{cases} e^{h(\mu)-1} - 1, & z_t = 1 \\ e^{h(z_t-1)-h(z_t)} - 1, & z_t \geq 2 \end{cases} \\ & \leq \frac{1}{2^{\mu+1}} \cdot \frac{1-c}{e} \cdot \begin{cases} h(\mu) - 1, & z_t = 1 \\ h(z_t - 1) - h(z_t), & z_t \geq 2 \end{cases}, \end{aligned} \quad (11.41)$$

where Eq. (11.41) holds by $e^x - 1 \leq x + x^2/2 \leq x/2$ for $-1 < x < 0$.

By combining Eqs. (11.37) and (11.41), we have

$$\begin{aligned} & \sum_{r \neq x_t} P(x_{t+1} = r \mid a(l) < x_t < b(l)) \cdot (e^{x_t - r} - 1) \\ & \leq \frac{ce}{1-ce} + \begin{cases} h(z_t+1) - h(z_t) + \frac{1}{2^{\mu+1}} \cdot \frac{1-c}{e} \cdot (h(\mu) - 1), & z_t = 1 \\ h(z_t+1) - h(z_t) + \frac{1}{2^{\mu+1}} \cdot \frac{1-c}{e} \cdot (h(z_t-1) - h(z_t)), & 1 < z_t < \mu \\ \frac{1}{2^{\mu+1}} \cdot \frac{1-c}{e} \cdot (h(z_t-1) - h(z_t)), & z_t = \mu \end{cases} \end{aligned} \quad (11.42)$$

If $z_t = 1$, $\frac{1-h(\mu)}{h(z_t+1)-h(z_t)} = \frac{d^\mu-d^{\mu-1}}{d^\mu-1} \cdot \frac{d^\mu-1}{d^{\mu-1}-d^{\mu-2}} = d$, and we have

$$\begin{aligned} & h(z_t+1) - h(z_t) + \frac{1}{2^{\mu+1}} \cdot \frac{1-c}{e} \cdot (h(\mu) - 1) \\ & = (h(z_t+1) - h(z_t)) \cdot \left(1 - d \cdot \frac{1}{2^{\mu+1}} \cdot \frac{1-c}{e}\right) \\ & \leq (h(z_t+1) - h(z_t)) \cdot (-1) \leq h(\mu - 1) - h(\mu). \end{aligned} \quad (11.43)$$

Note that $d = 2^{\mu+4}$ and $c = 1/(3 \cdot 2^{\mu(\mu+4)})$. If $1 < z_t < \mu$, $\frac{h(z_t)-h(z_t-1)}{h(z_t+1)-h(z_t)} = \frac{d^{\mu-z_t+1}-d^{\mu-z_t}}{d^{\mu-z_t}-d^{\mu-z_t-1}} = d$, and similarly we have

$$\begin{aligned} & h(z_t+1) - h(z_t) + \frac{1}{2^{\mu+1}} \cdot \frac{1-c}{e} \cdot (h(z_t-1) - h(z_t)) \\ & \leq h(z_t) - h(z_t+1) \leq h(\mu - 1) - h(\mu). \end{aligned} \quad (11.44)$$

If $z_t = \mu$, we have

$$\frac{1}{2^{\mu+1}} \cdot \frac{1-c}{e} \cdot (h(z_t-1) - h(z_t)) \leq \frac{2}{d} \cdot (h(\mu - 1) - h(\mu)). \quad (11.45)$$

Thus, Eq. (11.42) continues with

$$\begin{aligned} &\leq \frac{ce}{1-ce} + \frac{2}{d} \cdot (h(\mu-1) - h(\mu)) = \frac{1}{1/(ce)-1} + \frac{2}{d} \cdot \frac{1-d}{d^\mu-1} \\ &\leq \frac{1}{d^\mu-1} - \frac{3}{2} \cdot \frac{1}{d^\mu-1} = -\frac{1}{2(d^\mu-1)}, \end{aligned} \quad (11.46)$$

where Eq. (11.46) holds by $c = 1/(3d^\mu)$ and $d \geq 4$. This implies that Eq. (2.33) or equivalently Eq. (11.27), i.e., the condition of Theorem 2.7, holds with $p(l) = 2(d^\mu - 1)$.

Now, we need to analyze $D(l) = \max \{1, \mathbb{E}[e^{-\lambda(l) \cdot (x_{t+1} - b(l))} \mid x_t \geq b(l)]\} = \max \{1, \mathbb{E}[e^{b(l) - x_{t+1}} \mid x_t \geq b(l)]\}$ in Eq. (2.34). To derive an upper bound of $D(l)$, we only need to analyze $\mathbb{E}[e^{b(l) - x_{t+1}} \mid x_t \geq b(l)]$.

$$\begin{aligned} &\mathbb{E} \left[e^{b(l) - x_{t+1}} \mid x_t \geq b(l) \right] \quad (11.47) \\ &= \sum_{r \geq b(l)} P(y_{t+1} = r \mid x_t \geq b(l)) \cdot \mathbb{E} \left[e^{b(l) - x_{t+1}} \mid x_t \geq b(l), y_{t+1} = r \right] \\ &\quad + \sum_{r < b(l)} P(y_{t+1} = r \mid x_t \geq b(l)) \cdot \mathbb{E} \left[e^{b(l) - x_{t+1}} \mid x_t \geq b(l), y_{t+1} = r \right]. \end{aligned}$$

When $y_{t+1} = r \geq b(l)$, we have $b(l) - x_{t+1} = b(l) - y_{t+1} + h(z_{t+1}) \leq h(z_{t+1}) < 1$. We then consider the case of $y_{t+1} < b(l)$. Because $x_t = y_t - h(z_t) \geq b(l)$, we have $y_t \geq b(l) > y_{t+1}$, implying that $y_t \geq \lceil b(l) \rceil$ and $y_{t+1} \leq \lceil b(l) \rceil - 1$. To make $y_{t+1} = r \leq \lceil b(l) \rceil - 1$, it is necessary that a new solution s' with $|s'|_0 = r \leq \lceil b(l) \rceil - 1$ is generated by mutation. Let s denote the solution selected from the population P for mutation. Note that $|s|_0 \geq y_t \geq \lceil b(l) \rceil$. Then, $\forall r \leq \lceil b(l) \rceil - 1 : P(y_{t+1} = r \mid x_t \geq b(l)) \leq \sum_{s':|s'|_0=r} P_{\text{mut}}(s, s') \leq \sum_{s':|s'|_0=r} P_{\text{mut}}(s^{\lceil b(l) \rceil}, s') \leq (\lceil b(l) \rceil / n)^{\lceil b(l) \rceil - r} \leq (\lceil b(l) \rceil / n)^{\lceil b(l) \rceil - r}$. Furthermore, for $y_{t+1} < y_t$, it must hold that $z_{t+1} = 1$, and thus, $b(l) - x_{t+1} = b(l) - y_{t+1} + h(z_{t+1}) = b(l) - y_{t+1}$. Thus, Eq. (11.47) continues with

$$\begin{aligned} &\leq e + \sum_{r \leq \lceil b(l) \rceil - 1} \left(\frac{\lceil b(l) \rceil}{n} \right)^{\lceil b(l) \rceil - r} \cdot e^{b(l) - r} \\ &\leq e + \sum_{j=1}^{\lceil b(l) \rceil} \left(\frac{\lceil b(l) \rceil}{n} \right)^j \cdot e^j \leq e + \frac{e^{\lceil b(l) \rceil}/n}{1 - e^{\lceil b(l) \rceil}/n} \\ &= e + \frac{1}{n/(e^{\lceil b(l) \rceil}) - 1} \leq e + \frac{1}{1/(ce) - 1} \leq e + 1, \end{aligned} \quad (11.48)$$

where Eq. (11.48) holds by $\lceil b(l) \rceil \leq b(l) + 1 = cn$ and $c = 1/(3d^\mu)$. Thus,

$$D(l) = \max \left\{ 1, \mathbb{E} \left[e^{b(l) - x_{t+1}} \mid x_t \geq b(l) \right] \right\} \leq e + 1. \quad (11.49)$$

Let $L(l) = e^{cn/2}$ in Theorem 2.7. Note that $c = 1/(3d^\mu) = 1/(3 \cdot 2^{\mu(\mu+4)}) = \Theta(1)$ for $\mu = O(1)$. By Theorem 2.7, we have

$$\begin{aligned} \Pr(T(l) \leq e^{cn/2} \mid x_0 \geq b(l)) &\leq e^{1-cn} \cdot e^{cn/2} \cdot (e+1) \cdot 2(d^\mu - 1) \\ &= e^{-\Omega(n)}. \end{aligned} \quad (11.50)$$

By the Chernoff bound, for any s chosen from $\{0,1\}^n$ uniformly at random, $\Pr(|s|_0 < cn) = e^{-\Omega(n)}$. By the union bound, $\Pr(y_0 < cn) \leq \mu \cdot e^{-\Omega(n)} = e^{-\Omega(n)}$, implying $\Pr(x_0 < b(l)) = \Pr(y_0 - h(z_0) < b(l)) \leq \Pr(y_0 < b(l) + 1) = \Pr(y_0 < cn) = e^{-\Omega(n)}$. Thus, the expected running time is exponential. \square

11.2.2 Offspring Population

Next, we show the advantage of using offspring population over sampling on the robustness against noise. Particularly, we compare (1+1)-EA using sampling with (1+ λ)-EA in Algorithm 2.3 for solving the OneMax problem under reverse noise. As presented in Definition 11.2, reverse noise returns a reverse fitness $-f(s)$ with probability 1/2.

Definition 11.2 (Reverse Noise). Let $f^n(s)$ and $f(s)$ denote the noisy and true fitness of a solution s , respectively, then

$$f^n(s) = \begin{cases} f(s) & \text{with probability } 1/2; \\ -f(s) & \text{with probability } 1/2. \end{cases} \quad (11.51)$$

For OneMax under reverse noise, the distribution of $(f^n(s) - f^n(s'))$ for any two solutions s and s' is symmetric about 0. Thus, as we have found under symmetric noise, the algorithm behavior by using sampling is analogous to a random walk, and thus the optimization is inefficient. The proof of Theorem 11.5 is similar to that of Theorem 11.2, proving the exponential running time required by (1+1)-EA using sampling to solve OneMax under symmetric noise.

Theorem 11.5. For (1+1)-EA solving OneMax under reverse noise, if using sampling, the expected running time is exponential.

Proof. The proof can be finished in the same way as that of Theorem 11.2. The proof of Theorem 11.2 applies the simplified negative drift analysis approach in Theorem 2.5, and analyzes the drift $\mathbb{E}[x_t - x_{t+1} \mid x_t = i]$ by dividing it into two parts: E^+ and E^- . In the proof of Theorem 11.2, the analysis of E^+ does not rely on the noise model, and thus, the upper bound $E^+ \leq i/n$ still holds here. For the lower bound $(n-i)/(2en)$ of E^- , it relies on the property that for any two solutions s with $|s|_0 = i$ and s' with $|s'|_0 = i+1$, the distribution of $(f^n(s) - f^n(s'))$ is symmetric about 0. By the definition of reverse noise, $f^n(s) - f^n(s')$ can be $2i+1-2n, -1, 1$ and $2n-2i-1$, each with probability 1/4; thus its distribution is still symmetric about 0. Then, it holds that $E^- \geq (n-i)/(2en)$. According to Theorem 2.5, the expected running time is exponential. \square

By using offspring population, the probability of losing the current fitness becomes very small. This is because a fair number of offspring solutions with fitness no worse than the current fitness will be generated with a high probability in the reproduction of each iteration of $(1+\lambda)$ -EA, and the current fitness becomes worse only when all these good offspring solutions and the parent solution are evaluated incorrectly, the probability of which can be very small by using a logarithmic offspring population size. Thus, using offspring population can lead to efficient optimization, as shown in Theorem 11.6. Note that our finding for the effectiveness of using offspring population is consistent with that in [Gießen and Kötzing, 2016].

Theorem 11.6. *For $(1+\lambda)$ -EA solving OneMax under reverse noise, if $\lambda = 8 \log n$, the expected running time is $O(n \log^2 n)$.*

Proof. We use Theorem 2.4 for the proof. Each state of the corresponding Markov chain $\{\xi_t\}_{t=0}^{+\infty}$ is just a solution here. In other words, ξ_t corresponds to the solution after running t iterations of $(1+\lambda)$ -EA. We construct the distance function as $\forall x \in \{0, 1\}^n : V(x) = |x|_0$. Assume that currently $|x|_0 = i$, where $1 \leq i \leq n$. To examine $\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) | \xi_t = x]$, we divide it into two parts as in the proof of Theorem 11.3; that is,

$$\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) | \xi_t = x] = E^+ - E^-, \quad (11.52)$$

$$\text{where } E^+ = \sum_{\xi_{t+1}: V(\xi_{t+1}) < i} P(\xi_{t+1} | \xi_t = x) \cdot (i - V(\xi_{t+1})), \quad (11.53)$$

$$E^- = \sum_{\xi_{t+1}: V(\xi_{t+1}) > i} P(\xi_{t+1} | \xi_t = x) \cdot (V(\xi_{t+1}) - i). \quad (11.54)$$

For E^+ , as $V(\xi_{t+1}) < i$, we have $i - V(\xi_{t+1}) \geq 1$. Thus,

$$E^+ \geq \sum_{\xi_{t+1}: V(\xi_{t+1}) < i} P(\xi_{t+1} | \xi_t = x) = P(V(\xi_{t+1}) < i | \xi_t = x). \quad (11.55)$$

To make $V(\xi_{t+1}) < i$, it requires that at least one solution x' with $|x'|_0 < i$ is generated in reproduction and at least one of them is evaluated correctly. To generate a solution x' with $|x'|_0 < i$ by mutating x , it is sufficient that only one 0-bit of x is flipped, occurring with probability $(i/n) \cdot (1 - 1/n)^{n-1} \geq i/(en)$. Thus, in each iteration of $(1+\lambda)$ -EA, the probability of generating at least one offspring solution x' with $|x'|_0 < i$ is at least

$$1 - \left(1 - \frac{i}{en}\right)^\lambda \geq 1 - e^{-\lambda \cdot \frac{i}{en}} \geq 1 - \frac{1}{1 + \lambda \cdot \frac{i}{en}}. \quad (11.56)$$

If $\lambda \cdot i/(en) > 1$, $1 - (1 - i/(en))^\lambda \geq 1/2$; otherwise, $1 - (1 - i/(en))^\lambda \geq (\lambda \cdot i/(en))/(1 + \lambda \cdot i/(en)) \geq \lambda \cdot i/(2en)$. Thus, we have $1 - (1 - i/(en))^\lambda \geq \min\{1/2, \lambda \cdot i/(2en)\} = \min\{1/2, 4 \log n \cdot i/(en)\}$, where the equality holds

by $\lambda = 8 \log n$. Because each solution is evaluated correctly with probability $1/2$, $P(V(\xi_{t+1}) < i \mid \xi_t = x) \geq \min\{1/2, 4 \log n \cdot i/(en)\} \cdot (1/2)$. Thus,

$$E^+ \geq \min \left\{ \frac{1}{2}, \frac{4 \log n \cdot i}{en} \right\} \cdot \frac{1}{2} = \min \left\{ \frac{1}{4}, \frac{2 \log n \cdot i}{en} \right\} \geq \frac{i}{4n}. \quad (11.57)$$

For E^- , as $V(\xi_{t+1}) \leq n$, we have $V(\xi_{t+1}) - i \leq n - i$. Thus,

$$\begin{aligned} E^- &\leq \sum_{\xi_{t+1}: V(\xi_{t+1}) > i} P(\xi_{t+1} \mid \xi_t = x) \cdot (n - i) \\ &= (n - i) \cdot P(V(\xi_{t+1}) > i \mid \xi_t = x). \end{aligned} \quad (11.58)$$

Let $q = \sum_{x': |x'|_0 \leq i} P_{\text{mut}}(x, x')$ denote the probability of generating an offspring solution x' with at most i 0-bits by mutating x . Because it is sufficient that no bit is flipped or only one 0-bit is flipped in mutation, $q \geq (1 - 1/n)^n + (i/n) \cdot (1 - 1/n)^{n-1} \geq 1/e$. Now we analyze $P(V(\xi_{t+1}) > i \mid \xi_t = x)$. Assume that in the reproduction, exactly k offspring solutions with at most i 0-bits are generated, where $0 \leq k \leq \lambda$; this occurs with probability $\binom{\lambda}{k} \cdot q^k (1 - q)^{\lambda - k}$. If $k < \lambda$, the solution in the next generation has more than i 0-bits, i.e., $V(\xi_{t+1}) > i$, iff the fitness evaluation of these k offspring solutions and the parent solution x is all affected by noise, occurring with probability $1/2^{k+1}$. If $k = \lambda$, the solution in the next generation must have at most i 0-bits, i.e., $V(\xi_{t+1}) \leq i$. Thus, we have

$$\begin{aligned} P(V(\xi_{t+1}) > i \mid \xi_t = x) &= \sum_{k=0}^{\lambda-1} \binom{\lambda}{k} \cdot q^k (1 - q)^{\lambda - k} \cdot \frac{1}{2^{k+1}} \\ &\leq \frac{1}{2} \cdot \left(1 - \frac{q}{2}\right)^\lambda \leq \frac{1}{2} \cdot \left(1 - \frac{1}{2e}\right)^\lambda, \end{aligned} \quad (11.59)$$

where Eq. (11.59) holds by $q \geq 1/e$. Applying Eq. (11.59) to Eq. (11.58),

$$E^- \leq (n - i) \cdot \frac{1}{2} \cdot \left(1 - \frac{1}{2e}\right)^{8 \log n} \leq \frac{n - i}{2n^{2.3}} \leq \frac{1}{2n^{1.3}}, \quad (11.60)$$

where the base of the logarithm is 2.

By subtracting E^- from E^+ , we have

$$\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] \geq \frac{i}{4n} - \frac{1}{2n^{1.3}} \geq \frac{i}{5n} = \frac{1}{5n} \cdot V(x), \quad (11.61)$$

where the last inequality holds with sufficiently large n . Thus, by Theorem 2.4, we have

$$\mathbb{E}[\tau \mid \xi_0] \leq 5n(1 + \log n) = O(n \log n), \quad (11.62)$$

implying that the expected running time is $O(n \log^2 n)$, because it requires to re-evaluate the parent solution and evaluate the $\lambda = 8 \log n$ offspring solutions in each iteration. \square

Next, we prove that a constant offspring population size $\lambda = O(1)$ is insufficient for solving the noisy problem in polynomial time. This also implies that the effective value $\lambda = 8 \log n$ derived in the above theorem is nearly tight. From the proof, we can find that $\lambda = O(1)$ cannot guarantee a sufficiently small probability of losing the current fitness, and thus the optimization is inefficient.

Theorem 11.7. *For $(1+\lambda)$ -EA solving OneMax under reverse noise, if $\lambda = O(1)$, the expected running time is exponential.*

Proof. We use Theorem 2.5 for the proof. Let $x_t = |\mathbf{s}|_0$ be the number of 0-bits of the solution \mathbf{s} after t iterations of $(1+\lambda)$ -EA. We consider the interval $[0, n/(16(2e)^\lambda)]$, i.e., $a = 0$ and $b = n/(16(2e)^\lambda)$ in Theorem 2.5.

We analyze the drift $\mathbb{E}[x_t - x_{t+1} \mid x_t = i]$ for $1 \leq i < n/(16(2e)^\lambda)$. We divide the drift as

$$\mathbb{E}[x_t - x_{t+1} \mid x_t = i] = E^+ - E^-, \quad (11.63)$$

$$\text{where } E^+ = \sum_{j=0}^{i-1} P(x_{t+1} = j \mid x_t = i) \cdot (i - j), \quad (11.64)$$

$$E^- = \sum_{j=i+1}^n P(x_{t+1} = j \mid x_t = i) \cdot (j - i). \quad (11.65)$$

For E^+ , we need to derive an upper bound of $P(x_{t+1} = j \mid x_t = i)$ for $j < i$. Because $x_{t+1} = j$ implies that at least one offspring solution \mathbf{s}' with $|\mathbf{s}'|_0 = j$ is generated by mutating \mathbf{s} in reproduction, we have

$$\begin{aligned} P(x_{t+1} = j \mid x_t = i) &\leq 1 - \left(1 - \sum_{\mathbf{s}' : |\mathbf{s}'|_0 = j} P_{\text{mut}}(\mathbf{s}, \mathbf{s}') \right)^\lambda \\ &\leq \lambda \cdot \sum_{\mathbf{s}' : |\mathbf{s}'|_0 = j} P_{\text{mut}}(\mathbf{s}, \mathbf{s}'), \end{aligned} \quad (11.66)$$

implying

$$\begin{aligned} E^+ &\leq \sum_{j=0}^{i-1} \lambda \cdot \left(\sum_{\mathbf{s}' : |\mathbf{s}'|_0 = j} P_{\text{mut}}(\mathbf{s}, \mathbf{s}') \right) \cdot (i - j) \\ &= \lambda \cdot \sum_{\mathbf{s}' : |\mathbf{s}'|_0 < i} P_{\text{mut}}(\mathbf{s}, \mathbf{s}') \cdot (i - |\mathbf{s}'|_0) \leq \lambda \cdot \frac{i}{n}, \end{aligned} \quad (11.67)$$

where Eq. (11.67) holds by Eq. (11.18). For E^- , we have

$$E^- \geq \sum_{j=i+1}^n P(x_{t+1} = j \mid x_t = i) = P(x_{t+1} > i \mid x_t = i). \quad (11.68)$$

Let $q = \sum_{s':|s'|_0 \leq i} P_{\text{mut}}(s, s')$, where s is any solution with i 0-bits. Using the same analysis as Eq. (11.59), we have

$$\begin{aligned} P(x_{t+1} > i \mid x_t = i) &= \sum_{k=0}^{\lambda-1} \binom{\lambda}{k} \cdot q^k (1-q)^{\lambda-k} \cdot \frac{1}{2^{k+1}} \\ &= \frac{1}{2} \cdot \left(\left(1 - \frac{q}{2}\right)^\lambda - \left(\frac{q}{2}\right)^\lambda \right) = \frac{1}{2} \cdot \left(\left(\frac{q}{2} + 1 - q\right)^\lambda - \left(\frac{q}{2}\right)^\lambda \right) \\ &\geq \frac{1}{2} \cdot \lambda \cdot \left(\frac{q}{2}\right)^{\lambda-1} \cdot (1-q) \geq \lambda \cdot \frac{1}{8(2e)^\lambda}, \end{aligned} \quad (11.69)$$

where Eq. (11.69) holds by $q \geq 1/e$ and $1-q \geq \sum_{s':|s'|_0=i+1} P_{\text{mut}}(s, s') \geq (n-i)/(en) \geq 1/4$. Applying Eq. (11.69) to Eq. (11.68), we have

$$E^- \geq \lambda \cdot \frac{1}{8(2e)^\lambda}. \quad (11.70)$$

By subtracting E^- from E^+ , we have

$$\mathbb{E}[x_t - x_{t+1} \mid x_t = i] \leq \lambda \cdot \frac{i}{n} - \lambda \cdot \frac{1}{8(2e)^\lambda} \leq -\frac{\lambda}{16(2e)^\lambda}, \quad (11.71)$$

where the last inequality holds by $i < n/(16(2e)^\lambda)$. Thus, condition (1) of Theorem 2.5 holds with $\epsilon = \lambda/(16(2e)^\lambda)$, which is a constant for $\lambda = O(1)$.

To make $|x_{t+1} - x_t| \geq j$, it is necessary that at least one offspring solution generated by mutating s flips at least j bits of s . Let $p(j)$ denote the probability that at least j bits of s are flipped in mutation. It holds that $p(j) \leq \binom{n}{j} (1/n^j)$. Thus, we have

$$\begin{aligned} P(|x_{t+1} - x_t| \geq j \mid x_t \geq 1) &\leq 1 - (1 - p(j))^\lambda \\ &\leq \lambda \cdot p(j) \leq \lambda \cdot \binom{n}{j} \frac{1}{n^j} \leq 2\lambda \cdot \frac{1}{2^j}, \end{aligned} \quad (11.72)$$

implying that condition (2) of Theorem 2.5 holds with $\delta = 1$ and $r(l) = 2\lambda = O(1)$. Note that $l = b - a = n/(16(2e)^\lambda) = \Theta(n)$. Thus, by Theorem 2.5, the expected running time is exponential. \square

11.3 Summary

In this chapter, we first study the influence of population by analyzing the expected running time of $(\mu+\lambda)$ -EA for solving the UBoolean function class. We derive the lower bound $\Omega(n \log n + \mu + \lambda n \log \log n / \log n)$ by applying the switch analysis approach presented in Chapter 4. The results partially complete the running time comparison between $(\mu+\lambda)$ -EA and $(1+1)$ -EA

on OneMax and LeadingOnes, the two well-studied pseudo-Boolean problems. We can now conclude that when μ or λ is large, $(\mu+\lambda)$ -EA has a worse expected running time, suggesting that though often desired in practice, it should be careful to increase the population size.

Next, we show the robustness of population against noise by comparing it with sampling, an effective strategy to cope with noise as theoretically shown in Section 10.4. For solving the OneMax problem under symmetric noise, we prove that (1+1)-EA using sampling with any sample size always requires exponential time, whereas by using a parent population size $\mu = 3 \log n$, $(\mu+1)$ -EA can find the optimal solution in $O(n \log^3 n)$ time. For solving the OneMax problem under reverse noise, we prove that (1+1)-EA using sampling requires exponential time, whereas by using an offspring population size $\lambda = 8 \log n$, $(1+\lambda)$ -EA can find the optimal solution in $O(n \log^2 n)$ time. In both cases, we prove that the employed parent and offspring population sizes are almost tight. These results disclose that using population can enhance robustness against noise.



Constrained Optimization

Constrained optimization [Bertsekas, 1999] is to optimize an objective function with constraints that must be satisfied. It often appears in real-world optimization tasks and can be formally stated as follows. A solution is (in)feasible if it does (not) satisfy the constraints. The goal is to find a feasible solution maximizing the objective f .

Definition 12.1 (Constrained Optimization).

$$\begin{aligned} & \arg \max_{\mathbf{s} \in \mathcal{S}} \quad f(\mathbf{s}) \\ \text{s.t. } & g_i(\mathbf{s}) = 0 \quad \text{for } 1 \leq i \leq q, \\ & h_i(\mathbf{s}) \leq 0 \quad \text{for } q + 1 \leq i \leq m, \end{aligned} \tag{12.1}$$

where $f(\mathbf{s})$ is the objective function, $g_i(\mathbf{s})$ and $h_i(\mathbf{s})$ are the equality and inequality constraints, respectively.

Infeasible solutions were commonly considered to be worthless to the concerned problem, and thus conventional optimization techniques, such as the branch-and-bound algorithm and the A* algorithm, often avoid generating infeasible solutions and search within feasible regions only. In addition to the requirement that an optimization algorithm must output a feasible solution, keeping the search space as small as possible is another intuitive reason to search within feasible regions only.

Many approaches have been developed to enable EAs to handle constrained optimization problems [Coello Coello, 2002, Mezura-Montes and Coello Coello, 2005]. Among those approaches, infeasible solutions are also usually avoided, repaired or penalized. Approaches avoiding infeasible solutions ensure that no infeasible solutions will be generated during the evolutionary process, such as through homomorphous mapping [Koziel and Michalewicz, 1999]; approaches repairing infeasible solutions transform infeasible solutions into feasible ones through some heuristic rules, e.g., a binary coded infeasible solution for the knapsack problem can be

repaired to be feasible by twice scanning [Raidl, 1998]; approaches penalizing infeasible solutions try to transform the original constrained problem into an unconstrained one by changing the fitness function or the selection function, e.g., death penalty assigns infeasible solutions the worst fitness [Coello Coello, 2002], and superior penalty selects the feasible one in comparison between feasible and infeasible individuals [Deb, 2000].

While “keeping the search space small” is an intuitive reason to kick infeasible solutions away, it has been reported that the death penalty function method where infeasible solutions are unlikely to join the evolution process, is worse than “softer” penalty function methods [Coit and Smith, 1996, Michalewicz and Schoenauer, 1996] and other methods such as [Runarsson and Yao, 2000] where infeasible solutions have a chance to be involved. It has been conjectured [Coello Coello, 2002] that, when there are many disjointed feasible regions, infeasible solutions could bridge these regions, and hence might be helpful to the search. It has also been hypothesized that infeasible solutions can cause “shortcuts” [Rogers et al., 2006], although no significant support found. These empirical findings suggest that infeasible solutions might be useful in the evolutionary search.

Theoretically, it is unclear whether and when infeasible solutions are helpful. A few initial theoretical works have emerged. In [Zhou and He, 2007], the penalty coefficient of a penalty-based approach was studied; in [He and Zhou, 2007], the penalty-based approach was compared with the repair-based approach on the knapsack problem.

This chapter theoretically studies the problem that under what conditions exploring infeasible solutions can be helpful. We present sufficient as well as necessary conditions [Yu and Zhou, 2008b], for that an EA will reach an optimal solution faster when infeasible solutions are involved.

Next, this chapter theoretically examines the effectiveness of *Pareto optimization* [Coello Coello, 2002, Cai and Wang, 2006], a common strategy to exploit infeasible solutions in evolutionary search. The main idea is to reformulate the original constrained optimization problem into a bi-objective optimization problem by treating the constraint violation degree as an additional minimization objective, and then solve the problem by an MOEA. Through this method, both feasible and infeasible solutions will be maintained in the population and have some chance to reproduce new solutions. Its performance has only been empirically tested on some benchmark problems [Venkatraman and Yen, 2005, Cai and Wang, 2006], while theoretical understanding of Pareto optimization is underdeveloped.

We try to shed some light on the power of Pareto optimization. We present the efficiency comparison [Qian et al., 2015b] between Pareto optimization and the penalty function method on two constrained combinatorial optimization problems. Our results show that, on the P-solvable *minimum matroid optimization* problem, Pareto optimization uses less time to find an optimal solution; on the NP-hard *minimum cost coverage* problem, Pareto optimization is exponentially faster to find an approximate solution.

The rest of this chapter is organized as follows. Sections 12.1 and 12.2 study the usefulness of infeasible solutions and the effectiveness of Pareto optimization, respectively. Section 12.3 concludes this chapter.

12.1 Usefulness of Infeasible Solutions

In this section, we first present sufficient and necessary conditions for infeasible solutions to be helpful, and then apply to the constrained Trap problem, which has been studied in Chapter 3.

12.1.1 Conditions for Exploring Infeasible Solutions

Let superscripts of F and I be the notations for “feasible” and “infeasible”, respectively. The solution space \mathcal{S} consists of a feasible subspace \mathcal{S}^F and an infeasible one \mathcal{S}^I . \mathcal{S}^F contains an optimal solution, which is the goal of optimization. On the population level, given the population size m , we denote $\mathcal{X}^F = (\mathcal{S}^F)^m \subseteq \mathcal{X}$, where every population contains no infeasible solution, and $\mathcal{X}^I = \mathcal{X} \setminus \mathcal{X}^F$, where every population contains at least one infeasible solution. We want to compare two EAs, i.e., EA^F and EA , where EA^F is restricted in feasible regions, whereas EA explores infeasible regions.

Note that there are many different ways to kick away as well as to involve infeasible solutions. To model EAs using different ways of dealing with infeasible solutions, we characterize the behaviors of EAs by characterizing their search region. Formally, EA^F searches in the subspace \mathcal{X}^F , whereas EA searches in the whole population space \mathcal{X} . In other words, EA is possible to generate a population in \mathcal{X}^I , which is impossible for EA^F .

For a Markov chain $\{\xi_t\}_{t=0}^{+\infty}$ modeling the process of an EA, we denote the one-step success probability of a state x as

$$\mathcal{G}_t(x) = P(\xi_{t+1} \in \mathcal{X}^* \mid \xi_t = x). \quad (12.2)$$

Note that a state x represents a population. For a set A of states, if for its two arbitrary members x, y it holds that $\mathcal{G}_t(x) = \mathcal{G}_t(y)$, we denote $\mathcal{G}_t(A) = \mathcal{G}_t(x)$ for arbitrary $x \in A$. It will be seen that $\mathcal{G}_t(\cdot)$ is a critical measure of *goodness* of states (populations). A sufficient condition and a necessary condition for the usefulness of infeasible solutions are presented as follows.

Theorem 12.1. *For two absorbing Markov chains $\{\xi_t^F\}_{t=0}^{+\infty}$ and $\{\xi_t^I\}_{t=0}^{+\infty}$ corresponding to the processes of EA^F and EA , respectively, suppose $P(\xi_0^F \in \mathcal{X}^*) = P(\xi_0 \in \mathcal{X}^*)$. If it satisfies*

$$\forall t \geq 0 : \sum_{x \in \mathcal{X}^F \setminus \mathcal{X}^*} \mathcal{G}_t(x) \tilde{p}(\xi_t^F = x) \leq \sum_{x \in \mathcal{X} \setminus \mathcal{X}^*} \mathcal{G}_t(x) \tilde{p}(\xi_t = x), \quad (12.3)$$

then

$$\mathbb{E}[\tau^F \mid \xi_0^F \sim \pi_0^F] \geq \mathbb{E}[\tau \mid \xi_0 \sim \pi_0], \quad (12.4)$$

where

$$\tilde{p}(\xi_t^F = x) = \frac{P(\xi_t^F = x)}{1 - P(\xi_t^F \in \mathcal{X}^*)}, \quad (12.5)$$

$$\tilde{p}(\xi_t = x) = \frac{P(\xi_t = x)}{1 - P(\xi_t \in \mathcal{X}^*)}. \quad (12.6)$$

Proof. First, denote α_t as

$$\alpha_t = \sum_{x \in \mathcal{X}^F \setminus \mathcal{X}^*} \mathcal{G}_t(x) \tilde{p}(\xi_t^F = x), \quad (12.7)$$

and denote β_t as

$$\beta_t = \sum_{x \in \mathcal{X} \setminus \mathcal{X}^*} \mathcal{G}_t(x) \tilde{p}(\xi_t = x). \quad (12.8)$$

At time $t = 0$, by the assumption we have

$$P(\xi_0^F \in \mathcal{X}^*) = P(\xi_0 \in \mathcal{X}^*), \quad (12.9)$$

which satisfies $P(\xi_0^F \in \mathcal{X}^*) \leq P(\xi_0 \in \mathcal{X}^*)$.

Inductively hypothesize that $P(\xi_t^F \in \mathcal{X}^*) \leq P(\xi_t \in \mathcal{X}^*)$ at time t . Because the EAs are modeled by absorbing Markov chains, we have

$$P(\xi_{t+1}^F \in \mathcal{X}^*) = P(\xi_t^F \in \mathcal{X}^*) + \alpha_t(1 - P(\xi_t^F \in \mathcal{X}^*)), \quad (12.10)$$

$$P(\xi_{t+1} \in \mathcal{X}^*) = P(\xi_t \in \mathcal{X}^*) + \beta_t(1 - P(\xi_t \in \mathcal{X}^*)). \quad (12.11)$$

Let $\Delta_1 = P(\xi_t \in \mathcal{X}^*) - P(\xi_t^F \in \mathcal{X}^*)$ and $\Delta_2 = \beta_t - \alpha_t$. We have $\Delta_1 \geq 0$ by inductive hypothesis, and $\Delta_2 \geq 0$ by the condition, i.e., Eq. (12.3). Thus,

$$\begin{aligned} P(\xi_{t+1} \in \mathcal{X}^*) - P(\xi_{t+1}^F \in \mathcal{X}^*) &= \Delta_2(1 - \Delta_1 - P(\xi_t^F \in \mathcal{X}^*)) + \Delta_1(1 - \alpha_t) \\ &= \Delta_2(1 - P(\xi_t \in \mathcal{X}^*)) + \Delta_1(1 - \alpha_t) \\ &\geq 0. \end{aligned} \quad (12.12)$$

Therefore, we have $\forall t \geq 0 : P(\xi_t^F \in \mathcal{X}^*) \leq P(\xi_t \in \mathcal{X}^*)$. According to the definition of first hitting time in Definition 2.11, it can be rewritten as

$$\forall t \geq 0 : P(\tau^F \leq t) \leq P(\tau \leq t). \quad (12.13)$$

Considering $\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] = \sum_{t=0}^{+\infty} (1 - P(\tau \leq t))$, we directly have

$$\mathbb{E}[\tau^F \mid \xi_0^F \sim \pi_0^F] \geq \mathbb{E}[\tau \mid \xi_0 \sim \pi_0], \quad (12.14)$$

implying that, in expectation, EA^F requires more time than EA to find an optimal solution. \square

Theorem 12.2. For two absorbing Markov chains $\{\xi_t^F\}_{t=0}^{+\infty}$ and $\{\xi_t\}_{t=0}^{+\infty}$ corresponding to the processes of EA^F and EA , respectively, suppose $P(\xi_0^F \in \mathcal{X}^*) = P(\xi_0 \in \mathcal{X}^*)$. If

$$\mathbb{E}[\tau^F \mid \xi_0^F \sim \pi_0^F] \geq \mathbb{E}[\tau \mid \xi_0 \sim \pi_0], \quad (12.15)$$

there must exist $t \geq 0$ such that

$$\sum_{x \in \mathcal{X}^F \setminus \mathcal{X}^*} \mathcal{G}_t(x) \tilde{p}(\xi_t^F = x) \leq \sum_{x \in \mathcal{X} \setminus \mathcal{X}^*} \mathcal{G}_t(x) \tilde{p}(\xi_t = x), \quad (12.16)$$

where

$$\tilde{p}(\xi_t^F = x) = \frac{P(\xi_t^F = x)}{1 - P(\xi_t^F \in \mathcal{X}^*)}, \quad (12.17)$$

$$\tilde{p}(\xi_t = x) = \frac{P(\xi_t = x)}{1 - P(\xi_t \in \mathcal{X}^*)}. \quad (12.18)$$

Proof. The proof can be accomplished by switching the directions of inequalities in Eqs. (12.3) and (12.4). \square

To have an intuitive understanding of the theorems, we focus on Eq. (12.3). The term $\mathcal{G}_t(x)$ is the success probability from a population, which can be viewed as the goodness of the population, and the term \tilde{p} measures the probability of the EA staying at a population given that the population is not optimal. Therefore, the sum of $\mathcal{G}_t(x) \cdot \tilde{p}(x)$ over the population space indicates the average goodness of the state of the EA. Because $\mathcal{X} = \mathcal{X}^F \cup \mathcal{X}^I$, Eq. (12.3) can be rewritten as

$$\begin{aligned} & \sum_{x \in \mathcal{X}^F \setminus \mathcal{X}^*} \mathcal{G}_t(x) \tilde{p}(\xi_t^F = x) - \sum_{x \in \mathcal{X}^F \setminus \mathcal{X}^*} \mathcal{G}_t(x) \tilde{p}(\xi_t = x) \\ & \leq \sum_{x \in \mathcal{X}^I \setminus \mathcal{X}^*} \mathcal{G}_t(x) \tilde{p}(\xi_t = x), \end{aligned} \quad (12.19)$$

where the left side can be explained as the loss of goodness in feasible regions as the EA pays for exploring infeasible regions, and the right side can be explained as the gain of goodness from exploring infeasible regions. Thus, briefly, if exploring infeasible regions leads to better solutions in terms of $\mathcal{G}_t(\cdot)$, it is worth carrying out.

Nevertheless, the above theorems are quite general, and the interpretability is hampered as the term \tilde{p} changes over time. We further derive two conditions with \tilde{p} eliminated; they are more restricted but more intuitive. For two variables u and v , denote $(u \mid C_1) \succeq (v \mid C_2)$ as $\exists i, \forall j \geq i, \forall k < i :$

$$\begin{aligned} & P(u = j \mid C_1) - P(v = j \mid C_2) \\ & \geq 0 \geq P(u = k \mid C_1) - P(v = k \mid C_2), \end{aligned} \quad (12.20)$$

which can be intuitively understood as that u is “closer” to the event corresponding to the largest number in the domain than v .

We then present Theorem 12.3, whose conditions can be interpreted as follows. Eq. (12.22) means that, the closer a population to the optimal populations, the closer its offspring populations to the optimal populations in expectation. This condition is therefore on the reproduction operator and the selection behavior, but not a property on the infeasible regions. This condition can be satisfied by simple mutation operators, such as one-bit mutation and bit-wise mutation. Eq. (12.23) means that, when infeasible solutions are involved, the EA produces better offspring populations in expectation, measured by $\mathcal{G}(\cdot)$. This can be achieved in three situations for the EA involving infeasible solutions, i.e., reproducing better feasible populations from an infeasible parent population, reproducing better infeasible populations from an infeasible parent population, and reproducing better infeasible populations from a feasible parent population. These three ways show the potential of helpfulness of exploring infeasible regions.

Theorem 12.3. *For two absorbing homogeneous Markov chains $\{\xi_t^F\}_{t=0}^{+\infty}$ and $\{\xi_t\}_{t=0}^{+\infty}$ corresponding to the processes of EA^F and EA, respectively, suppose $\forall i \in [m] : P(\xi_0^F \in \mathcal{X}_i) = P(\xi_0 \in \mathcal{X}_i)$, where $\{\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_m\}$ is a partition of the state space \mathcal{X} such that, for arbitrary two states x_1 and x_2 from the same subspace, $\mathcal{G}(x_1) = \mathcal{G}(x_2)$, and $\forall i > j : \mathcal{G}(\mathcal{X}_i) > \mathcal{G}(\mathcal{X}_j)$.*

$$\mathbb{E}[\tau^F \mid \xi_0^F \sim \pi_0^F] \geq \mathbb{E}[\tau \mid \xi_0 \sim \pi_0], \quad (12.21)$$

if the following two conditions hold:

- (1) $\forall x_1$ and x_2 satisfying $\mathcal{G}(x_1) > \mathcal{G}(x_2)$, it holds that

$$(\mathcal{G}(\xi_{t+1}) \mid \xi_t = x_1) \succeq (\mathcal{G}(\xi_{t+1}) \mid \xi_t = x_2); \quad (12.22)$$

- (2) $\forall x^F \in \mathcal{X}^F$ and $x \in \mathcal{X}$ satisfying $\mathcal{G}(x^F) = \mathcal{G}(x)$, it holds that

$$(\mathcal{G}(\xi_{t+1}) \mid \xi_t = x) \succeq (\mathcal{G}(\xi_{t+1}^F) \mid \xi_t^F = x^F). \quad (12.23)$$

The proof of Theorem 12.3 is provided in Appendix A.7. Similarly, we derive a condition under which search in infeasible regions should be avoided. Eq. (12.25) is a restriction of the reproduction operator and the selection behavior, while Eq. (12.26) means that, when exploiting infeasible regions, the offspring populations of the EA are led away from the optimal populations.

Theorem 12.4. *For two absorbing homogeneous Markov chains $\{\xi_t^F\}_{t=0}^{+\infty}$ and $\{\xi_t\}_{t=0}^{+\infty}$ corresponding to the processes of EA^F and EA, respectively, suppose $\forall i \in [m] : P(\xi_0^F \in \mathcal{X}_i) = P(\xi_0 \in \mathcal{X}_i)$, where $\{\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_m\}$ is a partition of the state space \mathcal{X} such that, for arbitrary two states x_1 and x_2 from the same subspace, $\mathcal{G}(x_1) = \mathcal{G}(x_2)$, and $\forall i > j : \mathcal{G}(\mathcal{X}_i) > \mathcal{G}(\mathcal{X}_j)$.*

$$\mathbb{E}[\tau^F \mid \xi_0^F \sim \pi_0^F] \leq \mathbb{E}[\tau \mid \xi_0 \sim \pi_0], \quad (12.24)$$

if the following two conditions hold:

(1) $\forall x_1$ and x_2 satisfying $\mathcal{G}(x_1) > \mathcal{G}(x_2)$, it holds that

$$(\mathcal{G}(\xi_{t+1}^F) \mid \xi_t^F = x_1) \succeq (\mathcal{G}(\xi_{t+1}^F) \mid \xi_t^F = x_2); \quad (12.25)$$

(2) $\forall x^F \in \mathcal{X}^F$ and $x \in \mathcal{X}$ satisfying $\mathcal{G}(x^F) = \mathcal{G}(x)$, it holds that

$$(\mathcal{G}(\xi_{t+1}^F) \mid \xi_t^F = x^F) \succeq (\mathcal{G}(\xi_{t+1}) \mid \xi_t = x). \quad (12.26)$$

12.1.2 Case Study

Next we apply the derived conditions to the case of EAs solving the constrained Trap problem in Definition 3.3. To allow exploring infeasible solutions, the following fitness function is used for maximization:

$$f(s) = - \left| c - \sum_{i=1}^n w_i s_i \right|, \quad (12.27)$$

which assigns fitness to feasible solutions as well as infeasible solutions.

There are many ways to handle infeasible solutions when the algorithm is allowed to explore infeasible regions. For the tidiness and simplicity, a simple mechanism is used to control the exploration of infeasible regions, i.e., introducing a boundary-across probability $p^I \in [0, 1]$. When selecting between a feasible parent solution and an infeasible offspring solution, or between an infeasible parent solution and a feasible offspring solution, the infeasible solution survives with the probability p^I .

Algorithm 12.1 presents (1+1)-EA with a boundary-across probability, called (1+1)-BapEA. Note that one-bit mutation rather than bit-wise mutation is employed here. When $p^I = 0$, (1+1)-BapEA never enters into infeasible regions; when $p^I = 1$, (1+1)-BapEA travels only in infeasible regions.

By applying Theorem 12.3, we find that exploiting infeasible solutions is desirable for (1+1)-BapEA solving the constrained Trap problem. During the running of (1+1)-BapEA, assume that the algorithm will stop once the offspring solution s' is optimal.

Theorem 12.5. *For solving the constrained Trap problem by (1+1)-BapEA, setting $p^I > 0$ is better than setting $p^I = 0$.*

Proof. Using one-bit mutation, one solution can be mutated to only solutions with Hamming distance 1 from the solution. Note that a state x is just a solution here. Therefore, the state space can be divided into three subspaces $\{\mathcal{X}_0, \mathcal{X}_1, \mathcal{X}_2\}$, such that $\mathcal{X}_2 = \mathcal{X}^*$ and

$$\mathcal{X}_0 = \{x \in \mathcal{X} \mid \|x - x^*\|_H > 1\}, \quad (12.28)$$

$$\mathcal{X}_1 = \{x \in \mathcal{X} \mid \|x - x^*\|_H = 1\}, \quad (12.29)$$

$$\mathcal{X}_2 = \{x \in \mathcal{X} \mid \|x - x^*\|_H = 0\}. \quad (12.30)$$

Algorithm 12.1 (1+1)-BapEA Algorithm

Input: pseudo-Boolean function $f : \{0, 1\}^n \rightarrow \mathbb{R}; p^I \in [0, 1]$

Output: solution from $\{0, 1\}^n$

Process:

```

1: let  $s$  = a solution uniformly and randomly selected from  $\{0, 1\}^n$ ;
2: while criterion is not met do
3:   apply one-bit mutation on  $s$  to generate  $s'$ ;
4:   if both  $s$  and  $s'$  are feasible or infeasible then
5:     if  $f(s') \geq f(s)$  then
6:        $s = s'$ 
7:     end if
8:   else
9:      $r$  = a value chosen from  $[0, 1]$  uniformly at random;
10:    if  $r < p^I$  then
11:       $s$  = the infeasible one between  $s$  and  $s'$ 
12:    else
13:       $s$  = the feasible one between  $s$  and  $s'$ 
14:    end if
15:  end if
16: end while
17: return  $s$ 
```

It can be seen that \mathcal{X}_2 contains the optimal solution $x^* = (0, \dots, 0, 1)$; \mathcal{X}_1 contains the solutions that have exactly two 1-bits, one of which is the last element, and a solution $x_0 = (0, \dots, 0, 0)$; \mathcal{X}_0 consists of the remaining solutions. Note that x_0 has the lowest fitness and no infeasible neighbors, implying $\forall x \neq x_0 : P(\xi_{t+1} = x_0 | \xi_t = x) = 0$. We denote EA as (1+1)-BapEA with $0 < p^I < 1$, and EA^F as (1+1)-BapEA with $p^I = 0$. We then have

$$\mathcal{G}(\mathcal{X}_0) = 0, \quad \mathcal{G}(\mathcal{X}_1) = \frac{1}{n}, \quad \mathcal{G}(\mathcal{X}_2) = 1. \quad (12.31)$$

For EA which involves infeasible solutions, assume that when x_0 occurs, it is replaced by $(1, 0, \dots, 0)$, which is a worse case because x_0 has $1/n$ chance to become x^* that is not for $(1, 0, \dots, 0)$. Considering that x_0 is replaced, we have

$$P(\xi_{t+1} \in \mathcal{X}_0 | \xi_t \in \mathcal{X}_0) \in \left\{ \frac{n-2}{n}, 1 - \frac{p^I}{n}, 1 \right\}, \quad P(\xi_{t+1} \in \mathcal{X}_1 | \xi_t \in \mathcal{X}_0) \in \left\{ \frac{2}{n}, \frac{p^I}{n}, 0 \right\},$$

$$P(\xi_{t+1} \in \mathcal{X}_2 | \xi_t \in \mathcal{X}_0) = 0, \quad (12.32)$$

$$P(\xi_{t+1} \in \mathcal{X}_0 | \xi_t \in \mathcal{X}_1) = \frac{1 - p^I}{n}, \quad P(\xi_{t+1} \in \mathcal{X}_1 | \xi_t \in \mathcal{X}_1) = \frac{n-2}{n} + \frac{p^I}{n},$$

$$P(\xi_{t+1} \in \mathcal{X}_2 | \xi_t \in \mathcal{X}_1) = \frac{1}{n}, \quad (12.33)$$

$$P(\xi_{t+1} \in \mathcal{X}_0 | \xi_t \in \mathcal{X}_2) = 0, \quad P(\xi_{t+1} \in \mathcal{X}_1 | \xi_t \in \mathcal{X}_2) = 0, \\ P(\xi_{t+1} \in \mathcal{X}_2 | \xi_t \in \mathcal{X}_2) = 1, \quad (12.34)$$

leading to

$$\forall i \in \{1, 2\} : P(\xi_{t+1} \in \mathcal{X}_i | \xi_t \in \mathcal{X}_1) - P(\xi_{t+1} \in \mathcal{X}_i | \xi_t \in \mathcal{X}_0) \geq 0, \quad (12.35)$$

$$P(\xi_{t+1} \in \mathcal{X}_0 | \xi_t \in \mathcal{X}_1) - P(\xi_{t+1} \in \mathcal{X}_0 | \xi_t \in \mathcal{X}_0) \leq 0, \quad (12.36)$$

$$\forall j \in \{0, 1\} : P(\xi_{t+1} \in \mathcal{X}_2 | \xi_t \in \mathcal{X}_2) - P(\xi_{t+1} \in \mathcal{X}_2 | \xi_t \in \mathcal{X}_j) \geq 0, \quad (12.37)$$

$$\begin{aligned} \forall i \in \{0, 1\} \forall j \in \{0, 1\} : \\ P(\xi_{t+1} \in \mathcal{X}_i | \xi_t \in \mathcal{X}_2) - P(\xi_{t+1} \in \mathcal{X}_i | \xi_t \in \mathcal{X}_j) \leq 0. \end{aligned} \quad (12.38)$$

Thus, the condition, i.e., Eq. (12.22), of Theorem 12.3 is satisfied.

For EA^F which kicks away infeasible solutions, $\mathcal{X}_1^F = \mathcal{X}_1 \cap \mathcal{X}^F = \{(0, \dots, 0, 0)\}$. Due to the influence of selection, we have

$$\begin{aligned} P(\xi_{t+1}^F \in \mathcal{X}_0^F | \xi_t^F \in \mathcal{X}_0^F) &= 1, & P(\xi_{t+1}^F \in \mathcal{X}_1^F | \xi_t^F \in \mathcal{X}_0^F) &= 0, \\ P(\xi_{t+1}^F \in \mathcal{X}_2^F | \xi_t^F \in \mathcal{X}_0^F) &= 0, \end{aligned} \quad (12.39)$$

$$\begin{aligned} P(\xi_{t+1}^F \in \mathcal{X}_0^F | \xi_t^F \in \mathcal{X}_1^F) &= \frac{n-1}{n}, & P(\xi_{t+1}^F \in \mathcal{X}_1^F | \xi_t^F \in \mathcal{X}_1^F) &= 0, \\ P(\xi_{t+1}^F \in \mathcal{X}_2^F | \xi_t^F \in \mathcal{X}_1^F) &= \frac{1}{n}, \end{aligned} \quad (12.40)$$

$$\begin{aligned} P(\xi_{t+1}^F \in \mathcal{X}_0^F | \xi_t^F \in \mathcal{X}_2^F) &= 0, & P(\xi_{t+1}^F \in \mathcal{X}_1^F | \xi_t^F \in \mathcal{X}_2^F) &= 0, \\ P(\xi_{t+1}^F \in \mathcal{X}_2^F | \xi_t^F \in \mathcal{X}_2^F) &= 1, \end{aligned} \quad (12.41)$$

leading to

$$\begin{aligned} \forall i \in \{0, 1, 2\}, \forall j \in \{1, 2\} : \\ P(\xi_{t+1} \in \mathcal{X}_j | \xi_t \in \mathcal{X}_i) - P(\xi_{t+1}^F \in \mathcal{X}_j^F | \xi_t^F \in \mathcal{X}_i^F) \geq 0, \end{aligned} \quad (12.42)$$

$$\begin{aligned} \forall i \in \{0, 1, 2\} : \\ P(\xi_{t+1} \in \mathcal{X}_0 | \xi_t \in \mathcal{X}_i) - P(\xi_{t+1}^F \in \mathcal{X}_0^F | \xi_t^F \in \mathcal{X}_i^F) \leq 0. \end{aligned} \quad (12.43)$$

Thus, the condition, i.e., Eq. (12.23), of Theorem 12.3 is satisfied.

By Theorem 12.3, $\mathbb{E}[\tau^F | \xi_0^F \sim \pi_0^F] \geq \mathbb{E}[\tau | \xi_0 \sim \pi_0]$, implying that it is helpful to enable (1+1)-BapEA to exploit infeasible regions by setting $p^I > 0$.

□

12.2 Effectiveness of Pareto Optimization

In this section, we theoretically study the effectiveness of Pareto optimization, a common strategy exploiting infeasible solutions. Note that the binary space $\{0, 1\}^n$ is considered here. By Pareto optimization [Coello Coello, 2002, Cai and Wang, 2006], the original constrained optimization problem in Definition 12.1 is reformulated into a bi-objective optimization problem

$$\arg \max_{\mathbf{s} \in \{0,1\}^n} \left(f(\mathbf{s}), - \sum_{i=1}^m f_i(\mathbf{s}) \right), \quad (12.44)$$

where f_i is the penalty term expressing a violation degree of the i -th constraint. A straightforward way to set f_i is

$$f_i(\mathbf{s}) = \begin{cases} |g_i(\mathbf{s})| & 1 \leq i \leq q; \\ \max\{0, h_i(\mathbf{s})\} & q + 1 \leq i \leq m. \end{cases} \quad (12.45)$$

Then, an MOEA is employed to solve this problem. Finally, the best feasible solution is selected from the generated non-dominated solution set. The general procedure of Pareto optimization is illustrated in Figure 12.1. By this bi-objective formulation, a solution has an objective vector rather than a scalar objective value. Though an infeasible solution is worse on the second dimension than a feasible one, it may be better on the first dimension. Thus, infeasible solutions can be incomparable with feasible ones, and can be maintained in the population and used to reproduce new solutions.

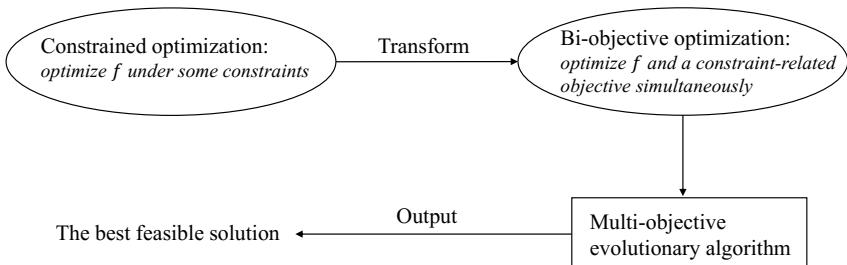


Figure 12.1. The general procedure of Pareto optimization.

To show the effectiveness of Pareto optimization, we compare it with the penalty function method [Ben Hadj-Alouane and Bean, 1997, Coello Coello, 2002], which has been widely employed to solve constrained optimization. For the optimization problem in Eq. (12.1), the penalty function method turns to solve an unconstrained optimization problem

$$\arg \max_{\mathbf{s} \in \{0,1\}^n} f(\mathbf{s}) - \lambda \sum_{i=1}^m f_i(\mathbf{s}), \quad (12.46)$$

where f is the objective in Eq. (12.1), λ is the penalty coefficient, and f_i is the constraint violation degree which can be set as Eq. (12.45). For λ , a static value has been shown to be robust [Homaifar et al., 1994, Michalewicz and Schoenauer, 1996]. Furthermore, Zhou and He [2007] proved that a large enough λ makes the penalty function method equivalent to the “superiority of feasible points” strategy [Deb, 2000] which prefers a smaller constraint

violation degree, and compares the objective value if having the same violation degree. Such a strategy ensures that a feasible solution is always better than an infeasible one, and thus the optimal solutions must be the same as that of the constrained optimization problem. A large enough static λ is considered in the analysis. After the transformation, an unconstrained optimization algorithm will be employed to solve this unconstrained problem.

For the fairness of comparison, GSEMO, i.e., Algorithm 2.5 using bit-wise mutation, is employed to solve the reformulated bi-objective problem for Pareto optimization, and (1+1)-EA in Algorithm 2.1 is employed to solve the reformulated single-objective unconstrained problem for the penalty function method. Their pseudo-codes are shown in Algorithms 12.2 and 12.3. It can be observed that these two algorithms use the same initialization strategy and both use bit-wise mutation to generate new solutions.

Algorithm 12.2 Pareto Optimization Method

Input: constrained optimization problem in Eq. (12.1)

Output: solution from $\{0, 1\}^n$

Process:

```

1: let  $g(\mathbf{s}) = (f(\mathbf{s}), -\sum_{i=1}^m f_i(\mathbf{s}))$ ;
2: let  $\mathbf{s}$  = a solution uniformly and randomly selected from  $\{0, 1\}^n$ ;
3: let  $P = \{\mathbf{s}\}$ ;
4: while criterion is not met do
5:   select a solution  $\mathbf{s}$  from  $P$  uniformly at random;
6:   apply bit-wise mutation on  $\mathbf{s}$  to generate  $\mathbf{s}'$ ;
7:   if  $\nexists \mathbf{z} \in P$  such that  $\mathbf{z} > \mathbf{s}'$  then
8:      $P = (P \setminus \{\mathbf{z} \in P \mid \mathbf{s}' \succeq \mathbf{z}\}) \cup \{\mathbf{s}'\}$ 
9:   end if
10: end while
11: return  $\mathbf{s} \in P$  with  $\sum_{i=1}^m f_i(\mathbf{s}) = 0$ 

```

Algorithm 12.3 Penalty Function Method

Input: constrained optimization problem in Eq. (12.1)

Output: solution from $\{0, 1\}^n$

Process:

```

1: let  $h(\mathbf{s}) = f(\mathbf{s}) - \lambda \sum_{i=1}^m f_i(\mathbf{s})$ ;
2: let  $\mathbf{s}$  = a solution uniformly and randomly selected from  $\{0, 1\}^n$ ;
3: while criterion is not met do
4:   apply bit-wise mutation on  $\mathbf{s}$  to generate  $\mathbf{s}'$ ;
5:   if  $h(\mathbf{s}') \geq h(\mathbf{s})$  then
6:      $\mathbf{s} = \mathbf{s}'$ 
7:   end if
8: end while
9: return  $\mathbf{s}$ 

```

Note that Pareto optimization is different from traditional multi-objective optimization [Deb et al., 2002]. The latter aims at finding a uniform approximation of the Pareto front, whereas Pareto optimization aims at finding the optimal solutions for the original constrained optimization problem.

We compare Pareto optimization with the penalty function method on two combinatorial optimization problems, i.e., minimum matroid optimization and minimum cost coverage. The former is P-solvable, and thus we use exact analysis, i.e., estimating the running time until an optimal solution is found. The latter is NP-hard, and thus we use approximate analysis, i.e., estimating the running time until finding an α -approximate solution which has the objective value at most $\alpha \cdot \text{OPT}$, where $\alpha > 1$.

As minimum matroid optimization and minimum cost coverage are large problem classes, worst-case running time [Cormen et al., 2001] is considered. By comparison, we will show the advantage of Pareto optimization over the penalty function method, and also that Pareto optimization can be better than the problem-specific algorithm, i.e., the greedy algorithm.

12.2.1 Minimum Matroid Optimization

First, consider the P-solvable minimum matroid optimization problem [Edmonds, 1971], which covers several combinatorial optimization problems, e.g., minimum spanning tree.

A matroid is a pair (V, S) , where V is a finite set and $S \subseteq 2^V$, satisfying

$$(1) \quad \emptyset \in S, \tag{12.47}$$

$$(2) \quad \forall A \subseteq B \in S : A \in S, \tag{12.48}$$

$$(3) \quad \forall A, B \in S, |A| > |B| : \exists v \in A \setminus B, B \cup \{v\} \in S. \tag{12.49}$$

The subsets in S are called *independent*. For any $A \subseteq V$, a maximum independent subset of A is called a *basis* of A ; the *rank* of A is the maximum cardinality of a basis of A , i.e., $r(A) = \max\{|B| \mid B \subseteq A, B \in S\}$. Note that for a matroid, all bases of A have the same cardinality.

Given a matroid (V, S) and a weight function $w : V \rightarrow \mathbb{N}^+$, the minimum matroid optimization problem is to find a basis of V with the minimum weight. Let $V = \{v_1, v_2, \dots, v_n\}$ and $w_i = w(v_i)$. Let $s \in \{0, 1\}^n$ represent a subset of V , where $s_i = 1$ means that v_i is selected. For notational convenience, s and its corresponding subset, i.e., $\{v_i \in V \mid s_i = 1\}$, will not be distinguished. The problem can then be formally stated as follows.

Definition 12.2 (Minimum Matroid Optimization). *Given a matroid (V, S) and a weight function $w : V \rightarrow \mathbb{N}^+$, to find a solution such that*

$$\arg \min_{s \in \{0,1\}^n} \left(w(s) = \sum_{i=1}^n w_i s_i \right) \quad \text{s.t.} \quad r(s) = r(V). \tag{12.50}$$

Note that, a rank oracle which computes the rank of a subset is polynomially equivalent to an independence oracle which tells whether a subset is independent [Korte and Vygen, 2012].

For Pareto optimization, considering $r(V) \geq r(s)$, the objective vector g is implemented as

$$g(s) = (-w(s), r(s) - r(V)). \quad (12.51)$$

Theorem 12.6 shows the running time bound, where $w_{\max} = \max\{w_i \mid 1 \leq i \leq n\}$ is the maximum element weight in V . The proof intuition is to divide the optimization process into two phases: (1) starts after initialization and finishes until finding the special solution 0^n ; (2) starts from 0^n and follows the greedy behavior [Edmonds, 1971] to find an optimal solution. The running time of phase (1) shown in Lemma 12.1 is derived by applying multiplicative drift analysis in Theorem 2.4.

Lemma 12.1. *For minimum matroid optimization, the expected running time of the Pareto optimization method for finding the special solution 0^n is $O(r(V)n(\log n + \log w_{\max}))$.*

Proof. We use Theorem 2.4 for the proof. Let a Markov chain $\{\xi_t\}_{t=0}^{+\infty}$ model the concerned process; that is, ξ_t corresponds to the population P after t iterations of Algorithm 12.2. We construct the distance function as $V(P) = \min\{w(s) \mid s \in P\}$, i.e., the minimum weight of solutions in the population P . Thus, $V(\xi_t) = 0$ implies that the solution 0^n has been found.

We analyze $\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t]$. Let s be the corresponding solution with $w(s) = V(\xi_t)$. $V(\xi_t)$ cannot increase, i.e., $V(\xi_{t+1}) \leq V(\xi_t)$, because a newly generated solution s' with a larger weight cannot weakly dominate s . Let P_{\max} denote the largest size of P during optimization. In the $(t+1)$ -th iteration, consider that s is selected in line 5 of Algorithm 12.2, occurring with probability at least $1/P_{\max}$. In line 6, let s^i denote the solution generated by flipping the i -th bit of s , i.e., s_i , and keeping the other bits unchanged, occurring with probability $(1/n)(1-1/n)^{n-1} \geq 1/(en)$. If $s_i = 1$, the generated solution s^i will enter into P as it has the smallest weight now, and $V(\xi_{t+1})$ will decrease to $V(\xi_t) - w_i$. Thus, we have

$$\begin{aligned} \mathbb{E}[V(\xi_{t+1})] &\leq \sum_{i:s_i=1} \frac{V(\xi_t) - w_i}{enP_{\max}} + \left(1 - \sum_{i:s_i=1} \frac{1}{enP_{\max}}\right) V(\xi_t) \\ &= V(\xi_t) - \frac{w(s)}{enP_{\max}} = \left(1 - \frac{1}{enP_{\max}}\right) V(\xi_t), \end{aligned} \quad (12.52)$$

implying $\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t] \geq V(\xi_t)/(enP_{\max})$.

The procedure of Algorithm 12.2 ensures that the solutions maintained in P must be incomparable, implying that there exists at most one solution in P for each value of one objective. Because the objective $r(s) - r(V) \in$

$\{0, -1, \dots, -r(V)\}$, $P_{\max} \leq r(V) + 1$. Note that $V(\xi_0) \leq nw_{\max}$ and $V_{\min} \geq 1$ by $w_i \in \mathbb{N}^+$. Thus, by Theorem 2.4, we have

$$\begin{aligned}\forall \xi_0 : \mathbb{E}[\tau \mid \xi_0] &\leq en(r(V) + 1)(1 + \log(nw_{\max})) \\ &= O(r(V)n(\log n + \log w_{\max})).\end{aligned}\quad (12.53)$$

□

Theorem 12.6. *For minimum matroid optimization, the expected running time of the Pareto optimization method for finding an optimal solution is $O(r(V)n(\log n + \log w_{\max} + r(V)))$.*

Proof. We analyze phase (2) after finding the special solution 0^n . Let $w^k = \min\{w(s) \mid r(s) = k\}$ denote the smallest weight in all solutions with rank k . w^k increases with k . The Pareto front of g is $\{(-w^k, k - r(V)) \mid 0 \leq k \leq r(V)\}$. We will show that starting from 0^n with $(0, -r(V))$, the algorithm can find the solutions corresponding to the Pareto front from $k = 0$ to $r(V)$ sequentially, where the last solution with $(-w^{r(V)}, 0)$ is just an optimal solution of Eq. (12.50) to find.

We first prove that $\forall s$ corresponding to $(-w^k, k - r(V))$, adding the element with the smallest weight which makes $r(s)$ increase by 1 will generate a solution s' corresponding to $(-w^{k+1}, k + 1 - r(V))$. Assume that $s = \{v_{i_1}, \dots, v_{i_k}\}$, $s' = s \cup \{v_{i_{k+1}}\}$, and there exists another solution $\hat{s} = \{v_{j_1}, \dots, v_{j_k}, v_{j_{k+1}}\}$ with rank $(k + 1)$ and $w(\hat{s}) < w(s')$. Assume that $w_{j_1} \leq \dots \leq w_{j_{k+1}}$. If $w_{j_{k+1}} \geq w_{i_{k+1}}$, $w(\hat{s}) - w_{j_{k+1}} < w(s') - w_{i_{k+1}} = w(s)$, contradicting with the fact that s has the smallest weight in all solutions with rank k . If $w_{j_{k+1}} < w_{i_{k+1}}$, then $\exists v \in \hat{s} \setminus s$, $s \cup \{v\}$ has rank $(k + 1)$ and $w(v) \leq w_{j_{k+1}} < w_{i_{k+1}}$, contradicting with the fact that $v_{i_{k+1}}$ is the element with the smallest weight which increases $r(s)$ by 1. Thus, our claim holds.

Assume that $\{(-w^k, k - r(V)) \mid 0 \leq k \leq i\}$ has been found. This is well defined because $i \geq 0$ after finding 0^n . Based on the above analysis, for finding $(-w^{i+1}, i + 1 - r(V))$ in one iteration, it is sufficient to select the solution corresponding to $(-w^i, i - r(V))$ in line 5 of Algorithm 12.2 and flip only the element with the smallest weight which increases its rank by 1 in line 6, occurring with probability at least $(1/P_{\max}) \cdot (1/n)(1 - 1/n)^{n-1} \geq 1/(en(r(V) + 1))$. The expected running time is $O(r(V)n)$ for finding $(-w^{i+1}, i + 1 - r(V))$. Because $r(V)$ such processes are sufficient to find an optimal solution with $(-w^{r(V)}, 0)$, the expected running time of phase (2) is $O(r^2(V)n)$.

By combining the two phases, we get the total expected running time $O(r(V)n(\log n + \log w_{\max} + r(V)))$. □

For the penalty function method, we first show that minimum spanning tree is an instance of minimum matroid optimization and then give a concrete minimum spanning tree example where the penalty function method requires more running time than Pareto optimization.

As introduced in Section 9.3.1, the minimum spanning tree problem is to find a connected subgraph with the minimum weight from an undirected connected graph. Let $s \in \{0, 1\}^m$ represent a subset of the edge set E , where $s_i = 1$ means that the i -th edge is selected; let $c(s)$ denote the number of connected components of the subgraph represented by s . In other words, the problem is to find a solution s minimizing the weight $w(s)$ while $c(s) = 1$. Let $V = E$ and $S = \{\text{subgraphs without cycles}\}$. Then, (V, S) is a matroid, and $c(s) + r(s) = r(V) + 1$. It can be verified that the minimum spanning tree problem is an instance of the minimum matroid optimization problem in Definition 12.2 with such a configuration. Note that $|V| = m$ here.

Neumann and Wegener [2007] have analyzed the running time of the penalty function method on a specific minimum spanning tree example, as shown in Lemma 12.2. Theorem 12.7 holds by $r(V) = n - 1$.

Lemma 12.2. [Neumann and Wegener, 2007] *The expected running time until the penalty function method finds a minimum spanning tree for an example graph with $m = \Theta(n^2)$ and $w_{\max} = 3n^2$ is $\Theta(m^2 \log n)$, where m and n denote the number of edges and nodes, respectively.*

Theorem 12.7. *There exists a minimum matroid optimization instance, where the expected running time of the penalty function method for finding an optimal solution is $\Theta(r^2(V)|V|(\log |V| + \log w_{\max}))$.*

By comparing Theorem 12.6 where $n = |V|$ with Theorem 12.7, it can be seen that, for finding an optimal solution of the P-solvable minimum matroid optimization problem, Pareto optimization is faster than the penalty function method by at least a factor of $\min\{\log |V| + \log w_{\max}, r(V)\}$.

12.2.2 Minimum Cost Coverage

Next, consider the NP-hard minimum cost coverage problem. A representative instance is the submodular set cover problem [Wolsey, 1982], which arises in many applications, e.g., social networks [Kempe et al., 2003] and sensor placement [Krause et al., 2008a].

Let $V = \{v_1, v_2, \dots, v_n\}$ be a finite set. A set function $f : 2^V \rightarrow \mathbb{R}$ is monotone and submodular iff $\forall A, B \subseteq V$, $f(A) \leq f(B) + \sum_{v \in A \setminus B} (f(B \cup \{v\}) - f(B))$ [Nemhauser et al., 1978]. Let $s \in \{0, 1\}^n$ represent a subset of V . Minimum cost coverage can be formally stated as follows.

Definition 12.3 (Minimum Cost Coverage). *Given a monotone submodular function $f : 2^V \rightarrow \mathbb{R}$, some value $q \leq f(V)$ and a weight function $w : V \rightarrow \mathbb{N}^+$, to find a solution such that*

$$\arg \min_{s \in \{0,1\}^n} \left(w(s) = \sum_{i=1}^n w_i s_i \right) \quad \text{s.t.} \quad f(s) \geq q. \quad (12.54)$$

We consider f to be non-negative. Because the rank of a matroid is a monotone submodular function, minimum matroid optimization is actually an instance of minimum cost coverage with $q = r(V)$. We analyze them separately because minimum matroid optimization is P-solvable and minimum cost coverage is generally NP-hard. Exact and approximate analyses are used, respectively.

For minimum cost coverage by Pareto optimization, the objective vector \mathbf{g} is implemented as

$$\mathbf{g}(\mathbf{s}) = (-w(\mathbf{s}), -\max\{0, q - f(\mathbf{s})\}). \quad (12.55)$$

The running time bound is shown in Theorem 12.8. Let c denote the minimum real number making $c \cdot q$ and $\forall \mathbf{s} (c \cdot f(\mathbf{s}))$ to be integer. Let N denote the number of distinct f values in $[0, q]$. The proof intuition is similar to that for Theorem 12.6 except that it follows the greedy behavior [Wolsey, 1982] to find an approximate solution.

Theorem 12.8. *For minimum cost coverage, the expected running time of the Pareto optimization method for finding an H_{cq} -approximate solution is $O(Nn(\log n + \log w_{\max} + N))$.*

Proof. Because the objective $-\max\{0, q - f(\mathbf{s})\}$ has $(N + 1)$ distinct values and the solutions in P are incomparable, the size of P is no larger than $N + 1$. Using the same proof as Lemma 12.1 except $P_{\max} \leq N + 1$ here, we can derive that the expected running time for finding 0^n is $O(Nn(\log n + \log w_{\max}))$.

Let $R_k = H_{cq} - H_{c(q-k)}$. Let \mathbf{s}^* denote an optimal solution of Eq. (12.54). Then, $w(\mathbf{s}^*) = \text{OPT}$. Let K_{\max} denote the maximum value of k such that $\exists \mathbf{s} \in P$ with $\min\{q, f(\mathbf{s})\} = k$ and $w(\mathbf{s}) \leq R_k \cdot \text{OPT}$; that is, $K_{\max} = \max\{k \mid \exists \mathbf{s} \in P : \min\{q, f(\mathbf{s})\} = k \wedge w(\mathbf{s}) \leq R_k \cdot \text{OPT}\}$. We are to analyze the expected running time until $K_{\max} = q$, implying that it finds an R_q (i.e., H_{cq})-approximate solution.

After finding 0^n , $K_{\max} \geq 0$. Assume that currently $K_{\max} = k < q$. Let \mathbf{s} denote the corresponding solution with the value k , i.e., $\min\{q, f(\mathbf{s})\} = k$ and $w(\mathbf{s}) \leq R_k \cdot \text{OPT}$. If \mathbf{s} is kept in P , K_{\max} will not decrease. If \mathbf{s} is deleted, by lines 7-9 of Algorithm 12.2, the newly generated solution \mathbf{s}' weakly dominating \mathbf{s} , i.e., no worse than \mathbf{s} on both the two objectives, must be included into P . Considering $\max\{0, q - f(\mathbf{s})\} = q - \min\{q, f(\mathbf{s})\}$, we have $\min\{q, f(\mathbf{s}')\} = k' \geq \min\{q, f(\mathbf{s})\} = k$ and $w(\mathbf{s}') \leq w(\mathbf{s})$. As R_k increases with k , we have $w(\mathbf{s}') \leq R_k \cdot \text{OPT} \leq R_{k'} \cdot \text{OPT}$. Thus, $K_{\max} \geq k'$, i.e., K_{\max} will not decrease.

Next we show that K_{\max} can increase by flipping a specific 0-bit of \mathbf{s} . Let $f'(\mathbf{s}) = \min\{q, f(\mathbf{s})\}$. Let \mathbf{s}^i denote the solution generated by flipping the i -th bit of \mathbf{s} . Let $I = \{i \in [n] \mid f'(\mathbf{s}^i) - f'(\mathbf{s}) > 0\}$ denote the 0-bit positions of \mathbf{s} where the flipping can generate a solution with a positive increment on f' . Let $\delta = \min\{w_i/(f'(\mathbf{s}^i) - f'(\mathbf{s})) \mid i \in I\}$. It holds that $\delta \leq \text{OPT}/(q - k)$. Otherwise, $\forall v_i \in \mathbf{s}^* \setminus \mathbf{s} : w_i > (f'(\mathbf{s}^i) - f'(\mathbf{s})) \cdot \text{OPT}/(q - k)$, implying

$\sum_{v_i \in s^* \setminus s} w_i > (\sum_{v_i \in s^* \setminus s} (f'(s^i) - f'(s))) \cdot \text{OPT}/(q-k)$. As f is monotone and submodular, f' is also monotone and submodular, and then $f'(s^*) - f'(s) \leq \sum_{v_i \in s^* \setminus s} (f'(s^i) - f'(s))$. Thus, $\sum_{v_i \in s^* \setminus s} w_i > (f'(s^*) - f'(s)) \cdot \text{OPT}/(q-k) = \text{OPT}$, contradicting with $\sum_{v_i \in s^* \setminus s} w_i \leq w(s^*) = \text{OPT}$. Thus, by selecting s in line 5 of Algorithm 12.2 and flipping only the 0-bit corresponding to δ in line 6, it can generate a new solution s' with $\min\{q, f(s')\} = k' > k$ and

$$w(s') \leq w(s) + (k' - k) \cdot \frac{\text{OPT}}{q - k} \leq R_{k'} \cdot \text{OPT}. \quad (12.56)$$

Once generated, s' will enter into P . Otherwise, there must exist a solution in P dominating s' which has a larger f' and a smaller w ; this implies that K_{\max} is larger than k , contradicting with the assumption $K_{\max} = k$. After including s' , K_{\max} increases from k to k' .

The probability of flipping a specific 0-bit of s is at least $(1/(N+1)) \cdot (1/n)(1 - 1/n)^{n-1} \geq 1/(en(N+1))$. Thus, the expected running time for such a step of increasing K_{\max} is at most $en(N+1)$. Because N such steps are sufficient to make $K_{\max} = q$, the expected running time of this phase is $O(N^2n)$. By combining the two phases, the expected running time of the whole process is $O(Nn(\log n + \log w_{\max} + N))$. \square

For the penalty function method, we first show that the set cover problem in Definition 6.5 with weights restricted to be positive integers is an instance of minimum cost coverage, and then give a concrete example where the penalty function method is less efficient than Pareto optimization.

By letting $f(s) = |\cup_{i:s_i=1} S_i|$, which is monotone and submodular, the set cover problem in Eq. (6.17) is an instance of minimum cost coverage in Definition 12.3 with $q = m$, because $f(s) \leq m$ and $|U| = m$. The parameters c and N in Theorem 12.8 satisfy that $c \leq 1$ and $N \leq m$ for set cover. Friedrich et al. [2010] have analyzed the running time of the penalty function method on a specific set cover example, as shown in Lemma 12.3. Theorem 12.9 holds by letting ϵ being a constant and $w_{\max} = 2^n$.

Lemma 12.3. [Friedrich et al., 2010] Let $\delta > 0$ be a constant and $n^{\delta-1} \leq \epsilon < 1/2$. The expected running time of the penalty function method on a set cover example with $m = \epsilon(1-\epsilon)n^2$ for finding an approximation better than $(1-\epsilon)w_{\max}/\epsilon$ is exponential with respect to n .

Theorem 12.9. There exists a minimum cost coverage instance, where the expected running time of the penalty function method for finding an H_{cq} -approximate solution is exponential with respect to n , N and $\log w_{\max}$.

By comparing Theorems 12.8 and 12.9, it can be seen that, for obtaining an H_{cq} -approximate solution of the NP-hard minimum cost coverage problem, the Pareto optimization method is exponentially faster than the penalty function method.

12.2.3 Pareto Optimization vs. Greedy Algorithm

For the above two problems, there exist greedy algorithms [Edmonds, 1971, Wolsey, 1982] with good performance. Thus, a natural question is whether Pareto optimization can be better than greedy algorithms. We give a positive answer by analyzing a concrete instance of set cover, also minimum cost coverage.

Example 12.1. As shown in Figure 12.2, the ground set U contains $m = k(k - 1)$ elements, and the collection V contains $n = (k + 1)(k - 1)$ subsets $\{S_1^*, \dots, S_{k-1}^*, S_{1,1}, \dots, S_{1,k}, \dots, S_{k-1,1}, \dots, S_{k-1,k}\}$ with weights $\forall i : w(S_i^*) = 1 + 1/k^2$ and $\forall i, j : w(S_{i,j}) = 1/j$.

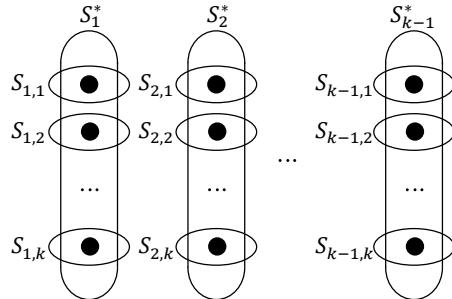


Figure 12.2. An example of the set cover problem.

The global optimal solution is $s_{global} = \{S_1^*, \dots, S_{k-1}^*\}$ with weight $(k - 1)(1 + 1/k^2)$, and $s_{local} = \{S_{1,1}, \dots, S_{1,k}, \dots, S_{k-1,1}, \dots, S_{k-1,k}\}$ is a local optimal solution with weight $(k - 1)H_k$. We show the performance of the greedy algorithm in Algorithm 6.2, the Pareto optimization method and the penalty function method on this example in Theorems 12.10-12.12, respectively. The greedy algorithm iteratively selects a set from V with the smallest ratio of its weight and the number of newly covered elements.

Theorem 12.10. *For Example 12.1, the greedy algorithm finds s_{local} .*

Proof. As in line 3 of Algorithm 6.2, let $r(S)$ denote the ratio between the weight of S and the number of newly covered elements by S . The greedy algorithm adds the set with the minimum ratio into the current solution at each step. At initialization, $\forall i : r(S_i^*) = (1 + 1/k^2)/k$, larger than the ratio $1/k$ of $S_{i,k}$, which is currently the smallest ratio. Thus, $S_{i,k}$ for all i will be added into the solution. After that, $\forall i : r(S_i^*) = (1 + 1/k^2)/(k - 1)$, larger than $1/(k - 1)$ of $S_{i,k-1}$. The greedy algorithm continues to choose non-optimal sets. Finally, all sets $S_{i,j}$ will be added into the solution, i.e., the local optimal solution s_{local} is found. \square

Theorem 12.11. *For Example 12.1, the Pareto optimization method finds s_{global} in $O(n^2 \log n)$ expected running time.*

Proof. The objective vector \mathbf{g} is implemented as $(-w(\mathbf{s}), |\cup_{i:s_i=1} S_i| - m)$. By using the same proof as Lemma 12.1 except that $w_{max} = 1 + 1/k^2$ and $P_{max} \leq m + 1$ in this example, we can derive an upper bound $O(mn \log n) = O(n^2 \log n)$ of the expected running time for finding 0^n .

Next we show that starting from 0^n , it can find the subset of $s_{global} = \{S_1^*, \dots, S_{k-1}^*\}$ with size from 0 to $(k-1)$ sequentially. Note that any subset of s_{global} is Pareto optimal, because the weight of covering k elements using sets only from $\{S_{i,j}\}$ is larger than $1 + 1/k^2$. After finding $\mathbf{s} \subseteq s_{global}$ with $|\mathbf{s}|_1 = i$, $\mathbf{s}' \subseteq s_{global}$ with $|\mathbf{s}'|_1 = i+1$ can be generated in one iteration with probability at least

$$\frac{1}{m+1} \cdot \frac{k-1-i}{n} \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{k-1-i}{e(m+1)n}, \quad (12.57)$$

because it is sufficient to select \mathbf{s} for mutation and flip only one 0-bit corresponding to any unselected S_i^* . Thus, the expected running time to find the optimal solution s_{global} is at most $\sum_{i=0}^{k-2} e(m+1)n/(k-1-i) = O(n^2 \log n)$. \square

Theorem 12.12. *For Example 12.1, the penalty function method finds s_{global} in $O(n^3 \log n)$ expected running time.*

Proof. Let u denote the number of uncovered elements of the current solution. It is clear that u will not increase. As each uncovered element has a corresponding $S_{i,j}$, flipping the corresponding bit of $S_{i,j}$ can decrease u by 1. Thus, u can decrease by 1 in one iteration with probability at least $(u/n)(1-1/n)^{n-1} \geq u/(en)$. For covering all elements, i.e., u decreasing to 0, the expected running time is at most $\sum_{u=m}^1 en/u = O(n \log n)$. Note that $u=0$ will be kept.

After $u=0$, we apply Theorem 2.4 to derive the running time for finding s_{global} . Let $V(\xi_t) = w(\xi_t) - OPT$, where $\xi_t = \mathbf{s}_t$ is the solution after t iterations of Algorithm 12.2 since $u=0$. $V(\xi_t) = 0$ implies that s_{global} is found. We analyze $\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) | \xi_t]$. It holds that $w(\mathbf{s}_{t+1}) \leq w(\mathbf{s}_t)$. For \mathbf{s}_t , there are three covering cases for each column of Figure 12.2. For case (1) which contains S_i^* and several $S_{i,j}$, by deleting one $S_{i,j}$, $w(\mathbf{s}_{t+1}) = w(\mathbf{s}_t) - 1/j$ with probability $(1/n)(1-1/n)^{n-1}$. For case (2) which contains all $S_{i,j}$ but not S_i^* , by including S_i^* and deleting $S_{i,1}$ and another $S_{i,j}$, $w(\mathbf{s}_{t+1}) = w(\mathbf{s}_t) - (1/j - 1/k^2)$ with probability $(1/n^3)(1-1/n)^{n-3}$; by including S_i^* and deleting $S_{i,1}$ and any two from $\{S_{i,2}, \dots, S_{i,k}\}$, $w(\mathbf{s}_{t+1}) < w(\mathbf{s}_t) - (k-2)/k^2$ with probability $((k-1)/n^4)(1-1/n)^{n-4}$. For case (3) which contains only S_i^* , it reaches the optimal case. By unifying these probabilities to a lower bound $(k-2)/(2e(k+1)n^3)$, the sum of improvements is just $w(\mathbf{s}_t) - OPT$. Thus, we have

$$\mathbb{E}[w(\mathbf{s}_{t+1}) \mid \mathbf{s}_t] \leq w(\mathbf{s}_t) - \frac{k-2}{2e(k+1)n^3}(w(\mathbf{s}_t) - \text{OPT}), \quad (12.58)$$

leading to

$$\begin{aligned} \mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t] &= w(\mathbf{s}_t) - \mathbb{E}[w(\mathbf{s}_{t+1}) \mid \mathbf{s}_t] \\ &\geq \frac{k-2}{2e(k+1)n^3} \cdot V(\xi_t). \end{aligned} \quad (12.59)$$

By Theorem 2.4 and considering that $V(\xi_0) < k^2$ and $V_{\min} \geq 1/k$, we have

$$\mathbb{E}[\tau \mid \xi_0] \leq \frac{2e(k+1)n^3}{k-2} \cdot (1 + 3 \log k) = O(n^3 \log n). \quad (12.60)$$

□

By comparing Theorems 12.10 to 12.12, it can be seen that both Pareto optimization and the penalty function method can be better than the greedy algorithm, among which Pareto optimization is the best.

12.3 Summary

This chapter theoretically studies how to deal with infeasible solutions when EAs are used for constrained optimization. We first present a theoretical study on the usefulness of infeasible solutions. We derive sufficient and necessary conditions to judge the usefulness of infeasible solutions in concrete problems. Next, we consider the question of how to exploit infeasible solutions well. We study the effectiveness of Pareto optimization, a common strategy exploiting infeasible solutions. By transforming the original constrained optimization problem into a bi-objective optimization problem which requires to optimize the original objective and minimize the constraint violation degree simultaneously, infeasible solutions are allowed to participate in the evolutionary process. Pareto optimization has shown its empirical advantages, though its theoretical ground is underdeveloped. We compare Pareto optimization with the widely used penalty function method on two large problem classes, the P-solvable minimum matroid optimization problem and the NP-hard minimum cost coverage problem. On both problems, Pareto optimization is proved to be more efficient. Moreover, Pareto optimization is shown to be better than the greedy algorithm. In Part IV, Pareto optimization will be employed to solve some constrained optimization problems in machine learning, exhibiting excellent performance both theoretically and empirically.

Part IV

LEARNING ALGORITHMS

Selective Ensemble

This chapter introduces how the theoretical advance of evolutionary optimization can help solve the problem of *selective ensemble*, also called *ensemble pruning*.

Ensemble methods [Zhou, 2012] are a kind of powerful machine learning approaches, which train and combine multiple individual learners for a learning task, as illustrated in Figure 13.1(a). They usually achieve state-of-the-art prediction performance, and thus have been widely applied to various tasks. Rather than combining all trained individual learners, selective ensemble [Rokach, 2010, Zhou, 2012] selects only a subset of individual learners to use, as illustrated in Figure 13.1(b). Reducing the compositing individual learners can save the storage space and accelerate the prediction speed. Furthermore, it has been shown that the selective ensemble can have better generalization performance than the whole ensemble [Zhou et al., 2002, Rokach, 2010].

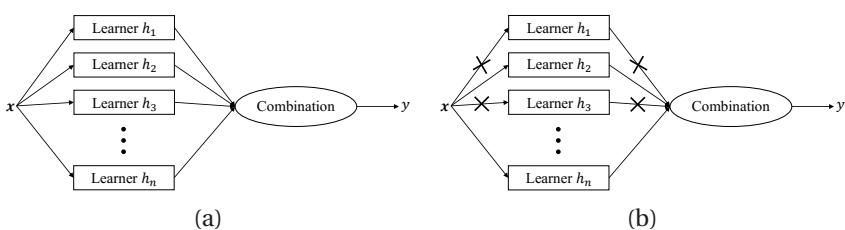


Figure 13.1. A common architecture of (a) ensemble and (b) selective ensemble.

Selective ensemble naturally bears two goals simultaneously, i.e., maximizing the generalization performance and minimizing the number of learners. When pushing to the limit, the two goals are conflicting, as overly fewer individual learners lead to poor performance. To achieve both good performance and a small ensemble size, previous selective ensemble ap-

proaches solve some objectives that mix the two goals. In Chapters 6 and 12, it has been revealed that, explicit consideration of every goal in a multi-objective formulation can be helpful for optimization. Thus, in this chapter, we consider the explicit bi-objective formulation of selective ensemble, and present the POSE algorithm [Qian et al., 2015a] based on Pareto optimization. POSE solves the bi-objective formulation of selective ensemble by an MOEA combined with a local search operator. We show theoretically that POSE is superior to the ordering-based selective ensemble methods in both performance and size. Empirical studies support that POSE is significantly better than many state-of-the-art algorithms. Finally, we apply POSE to the task of mobile human activity recognition.

The rest of this chapter is organized as follows. Section 13.1 introduces the selective ensemble problem. Sections 13.2 to 13.4 present the POSE algorithm, theoretical analysis and empirical study, respectively. Section 13.5 concludes this chapter.

13.1 Selective Ensemble

Given a set of trained individual learners, selective ensemble tries to select a small subset of individual learners to comprise the ensemble, without sacrificing or even improving the generalization performance contrasting to all-member ensembles. Zhou et al. [2002] have theoretically shown that ensemble some instead of all individual learners can be better. Another clear advantage of selective ensemble is reducing the storage cost and improving the efficiency, because fewer individual learners need to be stored and calculated in prediction.

Given a data set $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ and a set of n trained individual learners $H = \{h_i\}_{i=1}^n$, where $h_i : \mathcal{X} \rightarrow \mathcal{Y}$ maps the feature space \mathcal{X} to the label space \mathcal{Y} , let H_s denote a selective ensemble with the selector vector $s \in \{0, 1\}^n$, where $s_i = 1$ means that the individual learner h_i is selected. Selective ensemble tries to simultaneously optimize some objective f related to the generalization performance of H_s and minimize the size of H_s which is simply counted as $|s|_1$.

Selective ensemble usually refers to pruning ensembles generated by parallel methods such as Bagging. There are also studies [Margineantu and Dietterich, 1997, Tamon and Xiang, 2000] on pruning ensembles generated by sequential methods such as Boosting. All of them can be put under the umbrella of ensemble pruning. Note that early studies focused on sequential ensemble pruning, but it has been shown [Tamon and Xiang, 2000] that the pruning is intractable even to approximate, and often sacrifices generalization performance. Zhou et al. [2002] showed that pruning ensembles generated by parallel ensemble methods could lead to smaller ensembles with better generalization performance. Later, most ensemble pruning studies were devoted to parallel ensemble methods.

Existing selective ensemble techniques can be roughly categorized into two branches, i.e., the ordering-based and optimization-based methods. The ordering-based methods [Martínez-Muñoz et al., 2009, Partalas et al., 2012] commonly start from the empty set and then iteratively add an individual learner optimizing a certain objective. The sequence of being added into the selective ensemble gives an order of the individual learners, the front ones of which constitute the final ensemble. Algorithm 13.1 presents a common structure of the ordering-based selective ensemble methods, briefly called OSE, where the objective f is used to guide the search and an evaluation criterion, usually the validation error, is employed to select the final ensemble. Different ordering methods are mainly different on the choice of the objective, which can be minimizing the error [Margineantu and Dietterich, 1997], maximizing the diversity [Banfield et al., 2005], or combining the both [Li et al., 2012]. Many studies have shown that they can achieve a good selective ensemble efficiently [Martínez-Muñoz et al., 2009, Hernández-Lobato et al., 2011].

Algorithm 13.1 OSE Algorithm

Input: trained individual learners $H = \{h_i\}_{i=1}^n$; objective $f : 2^H \rightarrow \mathbb{R}$; criterion $eval$
Output: subset of H

Process:

- 1: let $H^S = \emptyset$ and $H^U = \{h_1, h_2, \dots, h_n\}$;
 - 2: **while** $H^U \neq \emptyset$ **do**
 - 3: $h^* = \arg \max_{h \in H^U} f(H^S \cup \{h\})$;
 - 4: $H^S = H^S \cup \{h^*\}$, and $H^U = H^U \setminus \{h^*\}$
 - 5: **end while**
 - 6: let $H^S = \{h_1^*, h_2^*, \dots, h_n^*\}$, where h_i^* is the learner added in the i -th iteration;
 - 7: let $k = \arg \min_{i \in [n]} eval(\{h_1^*, h_2^*, \dots, h_i^*\})$
 - 8: **return** $\{h_1^*, h_2^*, \dots, h_k^*\}$
-

The optimization-based methods formulate selective ensemble as a single-objective optimization problem which aims at finding a subset of individual learners with the best generalization performance. We briefly call them SOSE. Different optimization techniques have been employed, e.g., semi-definite programming [Zhang et al., 2006], quadratic programming [Li and Zhou, 2009] and heuristic optimization such as GA [Zhou et al., 2002] and artificial immune algorithms [Castro et al., 2005]. The heuristic methods use some trial-and-error style heuristics to search directly in the solution space. They are believed to be powerful, but lacking theoretical support. Moreover, empirical results have shown that the size of the selective ensemble by heuristic methods is often larger than that by ordering methods [Zhou et al., 2002, Li and Zhou, 2009]. Algorithm 13.2 gives an example of heuristic SOSE, which employs (1+1)-EA in Algorithm 2.1 to optimize some performance objective f directly.

Algorithm 13.2 SOSE Algorithm

Input: trained individual learners $H = \{h_i\}_{i=1}^n$; objective $f : 2^H \rightarrow \mathbb{R}$

Output: subset of H

Process:

- 1: let s = a solution uniformly and randomly selected from $\{0, 1\}^n$;
- 2: **while** criterion is not met **do**
- 3: apply bit-wise mutation on s to generate s' ;
- 4: **if** $f(s') \geq f(s)$ **then**
- 5: $s = s'$
- 6: **end if**
- 7: **end while**
- 8: **return** s

There are also studies on clustering-based pruning [Giacinto et al., 2000, Lazarevic and Obradovic, 2001]. The idea is to use clustering to identify a number of representative prototype individual learners to constitute the final ensemble. A clustering process is first employed to partition all individual learners into a number of groups, where individual learners in the same group behave similarly whereas different groups have large diversity. Then, the prototypes of clusters are put into the final ensemble.

13.2 The POSE Algorithm

We consider solving directly the bi-objective selective ensemble problem, formulated as

$$\arg \max_{s \in \{0, 1\}^n} (f(s), -|s|_1), \quad (13.1)$$

where $s \in \{0, 1\}^n$ represents a subset of individual learners, $f(s)$ is some objective related to the generalization performance of the selective ensemble H_s and $|s|_1$ is the number of selected learners. In the bi-objective formulation, the objective function, i.e., $(f(s), -|s|_1)$, gives any candidate solution s a vector rather than a scalar value. For example, a solution which results in value 0.2 of the employed f function and 10 classifiers will have the objective vector $(0.2, -10)$. Unlike single-objective optimization, the objective vector makes the comparison between two solutions less straightforward, because it is possible that one solution is better on the first dimension whereas the other is better on the second dimension. Thus, the domination relationship in Definition 1.2 is usually used for comparison in multi-objective optimization.

We present the Pareto Optimization algorithm for Selective Ensemble (POSE) in Algorithm 13.3. POSE employs a simple MOEA, GSEMO in Algorithm 2.5 with bit-wise mutation, to solve the bi-objective selective ensemble problem. It firstly generates a random solution, and puts it into the

Algorithm 13.3 POSE Algorithm

Input: trained individual learners $H = \{h_i\}_{i=1}^n$; objective $f : 2^H \rightarrow \mathbb{R}$; criterion $eval$
Output: subset of H
Process:

```

1: let  $g(\mathbf{s}) = (f(\mathbf{s}), -|\mathbf{s}|_1)$  be the bi-objective formulation;
2: let  $\mathbf{s}$  = a solution uniformly and randomly selected from  $\{0, 1\}^n$ ;
3: let  $P = \{\mathbf{s}\}$ ;
4: while criterion is not met do
5:   select a solution  $\mathbf{s}$  from  $P$  uniformly at random;
6:   apply bit-wise mutation on  $\mathbf{s}$  to generate  $\mathbf{s}'$ ;
7:   if  $\nexists \mathbf{z} \in P$  such that  $\mathbf{z} \succ \mathbf{s}'$  then
8:      $P = (P \setminus \{\mathbf{z} \in P \mid \mathbf{s}' \succeq \mathbf{z}\}) \cup \{\mathbf{s}'\}$ ;
9:      $Q = VDS(f, \mathbf{s}')$ ;
10:    for  $\mathbf{q} \in Q$ 
11:      if  $\nexists \mathbf{z} \in P$  such that  $\mathbf{z} \succ \mathbf{q}$  then
12:         $P = (P \setminus \{\mathbf{z} \in P \mid \mathbf{q} \succeq \mathbf{z}\}) \cup \{\mathbf{q}\}$ 
13:      end if
14:    end for
15:  end if
16: end while
17: return  $\arg \min_{\mathbf{s} \in P} eval(\mathbf{s})$ 

```

population P ; then follows a cycle to improve the solutions in P iteratively. In each iteration, a solution \mathbf{s} is selected randomly from P and then perturbed to generate a new solution \mathbf{s}' ; if \mathbf{s}' is not dominated by any solution in P , \mathbf{s}' is added into P and at the same time solutions in P that are weakly dominated by \mathbf{s}' get removed.

As POSE is inspired by EAs, which usually focus on global exploration but may utilize local information less well, a local search is incorporated into POSE to improve the quality of the new candidate solution in order to improve its efficiency. We employ the VDS algorithm [Lin and Kernighan, 1973], short for variable-depth search in Algorithm 13.4. It is known for the TSP as the Lin-Kernighan strategy. VDS performs a sequence of greedy local moves, each of which chooses the best local solution. To prevent loops in the moves, it keeps a set L to record the moving directions. VDS is applied to the newly included solution \mathbf{s}' in line 9 of Algorithm 13.3, and the generated solutions are then used to update P .

It is noticeable that the VDS subroutine considers only the objective f rather than both objectives. One may be curious about whether this will bias the search towards one objective. Because the selection in lines 7-8 and 10-14 of Algorithm 13.3 determines the search direction, VDS is used only to generate more potential candidate solutions for the selection in POSE, rather than to influence the search direction.

After a set of Pareto optimal solutions to the bi-objective formulation of selective ensemble has been solved, one final solution can be selected

Algorithm 13.4 VDS Subroutine

Input: pseudo-Boolean function $f : \{0, 1\}^n \rightarrow \mathbb{R}$; solution $\mathbf{s} \in \{0, 1\}^n$

Output: n solutions from $\{0, 1\}^n$

Process:

```

1: let  $Q = \emptyset$  and  $L = \emptyset$ ;
2: let  $N(\cdot)$  denote the set of neighbor solutions of a binary vector with Hamming
   distance 1;
3: while  $V_s = \{\mathbf{y} \in N(\mathbf{s}) \mid (y_i \neq s_i \Rightarrow i \notin L)\} \neq \emptyset$  do
4:   choose  $\mathbf{y} \in V_s$  with the maximum  $f$  value;
5:    $Q = Q \cup \{\mathbf{y}\}$ ;
6:    $L = L \cup \{i \mid y_i \neq s_i\}$ ;
7:    $\mathbf{s} = \mathbf{y}$ 
8: end while
9: return  $Q$ 

```

according to our preference, by optimizing some evaluation criterion in line 17 of Algorithm 13.3.

For the choice of the performance measure f , as the generalization performance is hard to be measured directly, an alternative way is to use the error directly on the training set or on a validation data set [Margineantu and Dietterich, 1997, Zhou et al., 2002, Caruana et al., 2004]. Other criteria have also been introduced to guide the selection, such as the diversity measures [Zhou, 2012]. A representative is the κ -statistics used in the Kappa algorithm [Banfield et al., 2005], which calculates the difference of two classifiers based on their disagreements on a data set. Many diversity measures have been proposed [Brown et al., 2005] and shown to be useful for selective ensemble [Li et al., 2012, Sun and Zhou, 2018]. A measure combining the data error and the diversity measure has been shown to lead to excellent performance [Li et al., 2012].

When selecting the final solution from the Pareto optimal set, the choice of the evaluation criterion $eval$ can be task-dependent.

13.3 Theoretical Analysis

In this section, we theoretically study the effectiveness of POSE by comparing it with OSE in Algorithm 13.1 and SOSE in Algorithm 13.2. We firstly show that, for any selective ensemble task, POSE can generate a solution at least as good as that by OSE. We then show that POSE is strictly better than OSE on some cases. Furthermore, we show that SOSE can be much worse than POSE/OSE.

13.3.1 POSE Can Do All of OSE

We prove in Theorem 13.1 that for any selective ensemble task, POSE can efficiently produce at least an equally good solution as that by OSE in both performance and size. The running time is counted as the number of selective ensemble evaluations, which is often the most time-consuming step. The intuition of the proof is as follows: first, POSE can efficiently find the special solution 0^n , i.e., none of the learners is selected; then POSE can apply VDS on 0^n to follow the process of OSE; thus POSE can produce a solution at least as good as that by OSE. Lemma 13.1 bounds the time for finding 0^n . Let P_{\max} denote the largest size of P in POSE during optimization.

Lemma 13.1. *The expected number of iterations of POSE for finding the special solution 0^n is $O(P_{\max}n \log n)$.*

Proof. Let $i = \min\{|s|_1 \mid s \in P\}$, i.e., the minimum number of 1-bits of solutions in P . i cannot increase because a solution with more 1-bits cannot weakly dominate a solution s with $|s|_1 = i$. Once a solution s' with $|s'| < i$ is generated, it will always be accepted, because it has the smallest size now and no other solution in P can dominate it. Thus, i can decrease by 1 in one iteration with probability at least $(1/P_{\max}) \cdot (i/n)(1 - 1/n)^{n-1}$, because it is sufficient to select a solution s with $|s|_1 = i$ from P , whose probability is at least $1/P_{\max}$, and flip only one 1-bit in mutation, whose probability is $(i/n)(1 - 1/n)^{n-1}$. The expected number of iterations, denoted by $\mathbb{E}[i]$, for decreasing i by 1 is at most $P_{\max}(n/i)(1/(1 - \frac{1}{n})^{n-1}) \leq eP_{\max}n/i$, where the inequality holds by $(1 - 1/n)^{n-1} \geq 1/e$. By summing up $\mathbb{E}[i]$, an upper bound $\sum_{i=1}^n \mathbb{E}[i] \leq \sum_{i=1}^n eP_{\max}n/i = O(P_{\max}n \log n)$ of the expected number of iterations for finding 0^n is derived. \square

Theorem 13.1. *For any objective and any size, POSE within $O(n^4 \log n)$ expected running time can find a solution weakly dominating that generated by OSE at the fixed size.*

Proof. Consider that the solution 0^n is generated by line 6 of Algorithm 13.3. As 0^n is Pareto optimal, it will go into the VDS process, which actually follows the process of OSE, because OSE starts from the empty set, i.e., 0^n , and iteratively adds one learner, i.e., changes one bit from 0 to 1, maximizing f . Denote s^* as the solution generated by OSE. Thus, the set Q of solutions output by VDS contains s^* , which will be used to update the population P in lines 10-14 of Algorithm 13.3, making P always contain a solution weakly dominating s^* .

Thus, we only need to analyze the expected number of iterations until generating 0^n by line 6. Consider two cases. If the initial solution is 0^n which will never be removed as it is Pareto optimal, 0^n can be regenerated by line 6 with probability at least $(1/P_{\max})(1 - 1/n)^n$, because it is sufficient to select 0^n from P , whose probability is at least $1/P_{\max}$, and flip no

bits in mutation, whose probability is $(1 - 1/n)^n$. In this case, the expected number of iterations is at most $P_{\max}/(1 - 1/n)^n \leq 2eP_{\max}$. Otherwise, the expected number of iterations is $O(P_{\max}n \log n)$ by Lemma 13.1. As any two solutions in P are incomparable, there exists at most one corresponding solution in P for each possible size, implying $P_{\max} \leq n + 1$ because the size $|s|_1 \in \{0, 1, \dots, n\}$. Thus, the expected number of iterations for generating 0^n is $O(n^2 \log n)$.

For each iteration of POSE, the running time is at most 1, i.e., evaluating s' , plus that of VDS. VDS takes n local moves because after every local move one index is added to L , and each local move performs at most n evaluations as $|V_s| \leq n$. Thus, the running time of one iteration is $O(n^2)$, implying that the expected time for finding a solution weakly dominating s^* is $O(n^4 \log n)$. \square

13.3.2 POSE Can Do Better Than OSE

Consider binary classification, i.e., $\mathcal{Y} = \{-1, 1\}$, combining individual classifiers by voting, and taking the validation set error, denoted as err , for the generalization performance measure as well as the evaluation criterion, i.e., $f(s) = -\text{err}(s)$ and $\text{eval}(s) = \text{err}(s)$. The validation set error is calculated as

$$\text{err}(s) = \frac{1}{m} \sum_{i=1}^m \mathbb{I}(H_s(\mathbf{x}_i) \neq y_i). \quad (13.2)$$

Let $\text{err}(s = 0^n) = +\infty$ to ensure that at least one classifier will be selected.

In this setting, a selective ensemble H_s is composed as

$$H_s(\mathbf{x}) = \arg \max_{y \in \mathcal{Y}} \sum_{i=1}^n s_i \cdot \mathbb{I}(h_i(\mathbf{x}) = y). \quad (13.3)$$

We define the difference of two classifiers as

$$\text{diff}(h_i, h_j) = \frac{1}{m} \sum_{k=1}^m \frac{1 - h_i(\mathbf{x}_k)h_j(\mathbf{x}_k)}{2}, \quad (13.4)$$

and the error of one classifier as

$$\text{err}(h_i) = \frac{1}{m} \sum_{k=1}^m \frac{1 - h_i(\mathbf{x}_k)y_k}{2}. \quad (13.5)$$

Both of them belong to $[0, 1]$. If $\text{diff}(h_i, h_j) = 1$ (or 0), h_i and h_j always make the opposite (or same) prediction; if $\text{err}(h_i) = 1$ (or 0), h_i always makes the wrong (or right) prediction.

Theorem 13.2 shows that, for the task type of selective ensemble described in Example 13.1, POSE can find the optimal selective ensemble

within polynomial time, whereas OSE finds only a sub-optimal one with larger error, or larger size, or both. For Example 13.1, the optimal selective ensemble consists of 3 individual classifiers, i.e., H' : each makes different mistakes, and the combination leads to zero error. The proof of Theorem 13.2 mainly states that OSE will first select the individual classifier h^* with the smallest error due to the greedy nature and can be misled by h^* , whereas POSE can first efficiently find $\{h^*\}$, and then VDS is applied to generate the optimal selective ensemble H' with a large probability.

Example 13.1.

$$\exists H' \subseteq H : (|H'| = 3 \wedge \forall g, h \in H' : \text{diff}(g, h) = \text{err}(g) + \text{err}(h)), \quad (13.6)$$

$$\exists h^* \in H \setminus H' : \begin{cases} \text{err}(h^*) < \min\{\text{err}(h) \mid h \in H'\}; \\ \forall h \in H' : \text{diff}(h, h^*) < \text{err}(h) + \text{err}(h^*), \end{cases} \quad (13.7)$$

$$\begin{aligned} \forall g \in H \setminus H' \setminus \{h^*\} : \text{err}(g) &> \max\{\text{err}(h) \mid h \in H'\} \\ \wedge \text{err}(g) + \text{err}(h^*) - \text{diff}(g, h^*) &> \min\{\text{err}(h) + \text{err}(h^*) - \text{diff}(h, h^*) \mid h \in H'\} \\ &+ \max\{\text{err}(h) + \text{err}(h^*) - \text{diff}(h, h^*) \mid h \in H'\}. \end{aligned} \quad (13.8)$$

Theorem 13.2. *For Example 13.1, OSE using Eq. (13.2) finds a solution with objective vector $(\leq 0, \leq -3)$ where the two equalities never hold simultaneously, whereas POSE finds a solution with objective vector $(0, -3)$ in $O(n^4 \log n)$ expected running time.*

Proof. Without loss of generality, assume that $h^* = h_1$, $H' = \{h_2, h_3, h_4\}$ and $\text{err}(h_2) = \min\{\text{err}(h) \mid h \in H'\}$. Let $d_i = (\text{err}(h_1) + \text{err}(h_i) - \text{diff}(h_1, h_i))/2$, i.e., the ratio of the same mistakes made by h_1 and h_i , which is greater than 0. Assume $d_3 \leq d_4$. Denote s^j as a Boolean vector of length j , i.e., $s^j \in \{0, 1\}^j$.

For OSE, it follows such an optimization path:

$$0^n \rightarrow 10^{n-1} \rightarrow 110^{n-2} \rightarrow 1110^{n-3} \rightarrow 11110^{n-4} \rightarrow \dots . \quad (13.9)$$

The corresponding err value changes as:

$$\begin{aligned} +\infty &\rightarrow \text{err}(h_1) \rightarrow (\text{err}(h_1) + \text{err}(h_2))/2 \rightarrow d_2 + d_3 \\ &\rightarrow (d_2 + d_3 + d_4)/2 \rightarrow \geq 0. \end{aligned} \quad (13.10)$$

The first “ \rightarrow ” of the path holds because h_1 has the smallest error; the second “ \rightarrow ” holds because the error of combining two classifiers is their average error and thus h_2 with the smallest error in the remaining classifiers is selected; the third “ \rightarrow ” holds by $\text{err}(1110^{n-3}) = d_2 + d_3 \leq \text{err}(11010^{n-4}) = d_2 + d_4$ and $\forall s \in \{1100s^{n-4} \mid |s^{n-4}|_1 = 1\} : \text{err}(s) > \min\{d_2, d_3, d_4\} + \max\{d_2, d_3, d_4\} \geq d_2 + d_3$; the fourth “ \rightarrow ” holds by $\forall s \in \{1110s^{n-4} \mid |s^{n-4}|_1 = 1\} : \text{err}(s) > (d_2 + d_3 + \min\{d_2, d_3, d_4\} + \max\{d_2, d_3, d_4\})/2 > (d_2 + d_3 + d_4)/2 = \text{err}(11110^{n-4})$. Because $(\text{err}(h_1) + \text{err}(h_2))/2 > \text{err}(h_1) >$

$d_2 + d_3$, OSE will output a solution with objective vector $(-d_2 - d_3, -3)$, $(-(d_2 + d_3 + d_4)/2, -4)$ or $(\leq 0, \leq -5)$.

For POSE, by Lemma 13.1, the expected number of iterations for finding 0^n is $O(P_{\max}n \log n)$. Then, 10^{n-1} will be generated in one iteration with probability at least $(1/P_{\max}) \cdot (1/n)(1 - 1/n)^{n-1} \geq 1/(enP_{\max})$, because it is sufficient to select 0^n from P and flip only the first 0-bit. Because 10^{n-1} is Pareto optimal, it will always keep in P . Once it is regenerated in line 6 of Algorithm 13.3, occurring with probability at least $(1/P_{\max}) \cdot (1 - 1/n)^n$, it will go into the VDS process, and with probability $\Omega(1/n)$ the solution path found by VDS on 10^{n-1} is:

$$10^{n-1} \rightarrow 110^{n-2} \rightarrow 1110^{n-3} \rightarrow 11110^{n-4} \rightarrow 01110^{n-4} \rightarrow \dots \quad (13.11)$$

The corresponding err value changes as:

$$\begin{aligned} \text{err}(h_1) &\rightarrow (\text{err}(h_1) + \text{err}(h_2))/2 \rightarrow d_2 + d_3 \\ &\rightarrow (d_2 + d_3 + d_4)/2 \rightarrow 0 \rightarrow \geq 0. \end{aligned} \quad (13.12)$$

Considering $P_{\max} \leq n+1$, the expected running time for finding the optimal solution 01110^{n-4} with objective vector $(0, -3)$ is $O(P_{\max}n \log n + P_{\max}n + P_{\max}n) \cdot O(n^2) = O(n^4 \log n)$. \square

13.3.3 POSE/OSE Can Do Better Than SOSE

Heuristic optimization methods like EAs have been employed for solving selective ensemble in a single-objective formulation. GASEN [Zhou et al., 2002] is probably the first such algorithm, and several other algorithms, e.g., artificial immune algorithms [Castro et al., 2005], have also been proposed. However, it was unknown theoretically how well these optimization methods can be. Algorithm 13.2 is a simple heuristic SOSE, which employs (1+1)-EA to optimize the performance objective f directly. We prove in Theorem 13.3 that, for the task type of selective ensemble described in Example 13.2, OSE, and thus POSE due to Theorem 13.1, can find the optimal selective ensemble efficiently, whereas SOSE in Algorithm 13.2 requires at least exponential running time.

Example 13.2.

$$\exists H' \subseteq H : (|H'| = n - 1 \wedge \forall g, h \in H' : \text{diff}(g, h) = 0), \quad (13.13)$$

$$\text{err}(H \setminus H') < \text{err}(h \in H'). \quad (13.14)$$

For Example 13.2, all individual classifiers make the same prediction except one which makes fewer mistakes. Without loss of generality, assume $H' = \{h_2, \dots, h_n\}$. Let $\text{err}(h_1) = c_1$ and $\text{err}(h \in H') = c_2$, where $c_1 < c_2$. Then, the err function of Eq. (13.2) is

$$\text{err}(\mathbf{s}) = \begin{cases} +\infty & \text{if } \mathbf{s} = 0^n; \\ c_1 & \text{if } \mathbf{s} = 10^{n-1}; \\ (c_1 + c_2)/2 & \text{if } |\mathbf{s}|_1 = 2 \wedge s_1 = 1; \\ c_2 & \text{otherwise.} \end{cases} \quad (13.15)$$

The proof intuition of Theorem 13.3 is that OSE can easily find the optimal solution 10^{n-1} by the first greedy step, whereas SOSE almost performs a random walk on a plateau and thus is inefficient. The analysis of SOSE uses the simplified negative drift analysis approach in Theorem 2.5.

Theorem 13.3. *For Example 13.2, OSE using Eq. (13.2) finds the optimal solution in $O(n^2)$ running time, whereas the running time of SOSE is at least $2^{\Omega(n)}$ with probability $1 - 2^{-\Omega(n)}$.*

Proof. For OSE, according to the objective $f(\mathbf{s}) = -\text{err}(\mathbf{s})$, it will follow such an optimization path:

$$0^n \rightarrow 10^{n-1} \rightarrow 1s^{n-1}, |\mathbf{s}^{n-1}|_1 = 1 \rightarrow \dots \quad (13.16)$$

The corresponding err value changes as:

$$+\infty \rightarrow c_1 \rightarrow (c_1 + c_2)/2 \rightarrow c_2. \quad (13.17)$$

Thus, it outputs the optimal solution 10^{n-1} . Its running time is fixed. In the i -th iteration of Algorithm 13.1 where $1 \leq i \leq n$, it requires to evaluate and compare $(n-i+1)$ selective ensembles, generated by combining the current ensemble with any of the $(n-i+1)$ unselected classifiers. Thus, the total running time is $\sum_{i=1}^n (n-i+1) = n(n+1)/2 = O(n^2)$.

For SOSE where a new solution is generated based on the current solution, we can model it by a Markov chain and use Theorem 2.5 for the proof. Let $a = 2$, $b = n/3$, and let $x_t = |\mathbf{s}|_1$ be the number of 1-bits of the solution after t iterations of Algorithm 13.2. Let $P_{\text{mut}}(i \rightarrow j)$ be the probability of generating $|\mathbf{s}'|_1 = j$ from $|\mathbf{s}|_1 = i$ by bit-wise mutation in line 3 of Algorithm 13.2. $\forall a < i < b$, we have

$$\begin{aligned} & \mathbb{E}[x_t - x_{t+1} \mid x_t = i] \\ &= i - \sum_{j=1}^n P_{\text{mut}}(i \rightarrow j) \cdot j - P_{\text{mut}}(i \rightarrow 0) \cdot i \end{aligned} \quad (13.18)$$

$$\leq i - \sum_{j=0}^n P_{\text{mut}}(i \rightarrow j) \cdot j = i - (n-i)\frac{1}{n} - i \left(1 - \frac{1}{n}\right) \leq -\frac{1}{3}, \quad (13.19)$$

where Eq. (13.18) holds because any solution except 0^n has at least the same objective value as a solution \mathbf{s} with $|\mathbf{s}|_1 = i > 2$, and then is always accepted; Eq. (13.19) holds by applying the linearity of expectation on $\sum_{j=0}^n P_{\text{mut}}(i \rightarrow j) \cdot j = \mathbb{E}[\sum_{i=1}^n s'_i]$, which is actually the expected number of 1-bits of the

new solution generated in line 3 of Algorithm 13.2. To make $|x_{t+1} - x_t| \geq j$, it is necessary to flip at least j bits. Then, we have

$$P(|x_{t+1} - x_t| \geq j \mid x_t > a) \leq \binom{n}{j} \frac{1}{n^j} \leq \frac{1}{j!} \leq \frac{2}{2^j}. \quad (13.20)$$

Eqs. (13.19) and (13.20) imply that the conditions of Theorem 2.5 hold with $\epsilon = -1/3$, $\delta = 1$ and $r(l) = 2$. Thus, we have $P(T \geq 2^{\Omega(n)}) = 1 - 2^{-\Omega(n)}$, where $T = \min\{t \geq 0 : x_t \leq 2 \mid x_0 \geq n/3\}$. By the Chernoff bound, $P(x_0 \geq n/3) = 1 - 2^{-\Omega(n)}$ due to the uniform distribution of initial solution. Thus, with probability $1 - 2^{-\Omega(n)}$, the running time for finding a solution s with $|s|_1 \leq 2$ is $2^{\Omega(n)}$, which is a running time lower bound for finding the optimal solution 10^{n-1} . \square

13.4 Empirical Study

In this section, we empirically compare the POSE algorithm with some state-of-the-art selective ensemble algorithms. We conduct experiments on 20 binary and 10 multiclass data sets,[†] on initial ensemble trained by Bagging [Breiman, 1996]. To assess each algorithm on each data set, the following process is repeated for 30 times. The data set is randomly and evenly split into three parts, each as the training set, the validation set and the test set. A Bagging of 100 C4.5 decision trees [Quinlan, 1993] is trained on the training set, then pruned by a selective ensemble algorithm using the validation set, and finally tested on the test set. Furthermore, we also apply POSE to the task of mobile human activity recognition using smartphones.

For POSE, the first goal is to minimize the validation error, which is also used as the evaluation criterion for the final ensemble selection. Two baselines are the full Bagging, which uses all individual classifiers, and the Best Individual (**BI**), which selects the best classifier according to the validation error. Five state-of-the-art ordering-based algorithms are compared, including Reduce-Error (**RE**) [Caruana et al., 2004], **Kappa** [Banfield et al., 2005], Complementarity (**CP**) [Martínez-Muñoz et al., 2009], Margin Distance (**MD**) [Martínez-Muñoz et al., 2009], and **DREP** [Li et al., 2012] algorithms. They mainly differ in their considered objectives relating to the generalization performance, and all use the validation error as the evaluation criterion for selecting the final ensemble. As a representative heuristic single-objective optimization algorithm, an **EA** [Bäck, 1996] is compared, which is similar to Algorithm 13.2 except that it generates and maintains n solutions in each iteration, minimizing the validation error. The parameter p for MD is set to 0.075 [Martínez-Muñoz et al., 2009], and the parameter ρ of DREP is selected from $\{0.2, 0.25, \dots, 0.5\}$ [Li et al., 2012]. The number of

[†]<http://archive.ics.uci.edu/ml/>.

iterations for POSE is set to $\lceil n^2 \log n \rceil$, i.e., the total number of evaluations $O(n^4 \log n)$ divided by the number of evaluations $O(n^2)$ in each iteration, suggested by Theorems 13.1 and 13.2. For the fairness of comparison, the number of iterations for EA is set to $\lceil n^3 \log n \rceil$ so that it costs the same number of evaluations as POSE.

13.4.1 Binary Classification

Some of the binary data sets are generated from the original multiclass data sets: from *letter*, *letter-ah* classifies “a” against “h”, and alike *letter-br* and *letter-oq*; *optdigits* classifies “0~4” against “5~9”; from *satimage*, *satimage-12v57* classifies labels “1” and “2” against “5” and “7”, and alike *satimage-2v5*; from *vehicle*, *vehicle-bo-vs* classifies “bus” and “opel” against “van” and “saab”, and alike *vehicle-b-v*.

Table 13.1 summarizes the results. As it is improper to have a single summarization criterion over multiple data sets and algorithms, we employ the *number of best*, the *number of direct win* that is a pairwise comparison followed by the *sign-test* [Demšar, 2006], the *t-test* for pairwise comparison on each data set, and the *rank* [Demšar, 2006]. POSE achieves the smallest test error (or size) on 60% (12/20) of the data sets, while the other algorithms are less than 35% (7/20). By the sign-test with confidence level 0.05, POSE is significantly better than all compared algorithms on size and all algorithms except Kappa and DREP on test error, indicated by the rows “POSE: Count of direct win”. Though the sign-test shows that Kappa and DREP are comparable, POSE is still better on more than 60% (12.5/20) data sets. From the *t-test* with confidence level 0.05, of which significant better and worse are indicated by “•” and “○”, respectively, POSE is never significantly worse than the compared algorithms on test error, and has only two losses on size, i.e., on *vehicle-bo-vs* to CP and DREP. We also compute the rank of each algorithm on each data set, which are stacked in Figure 13.2.

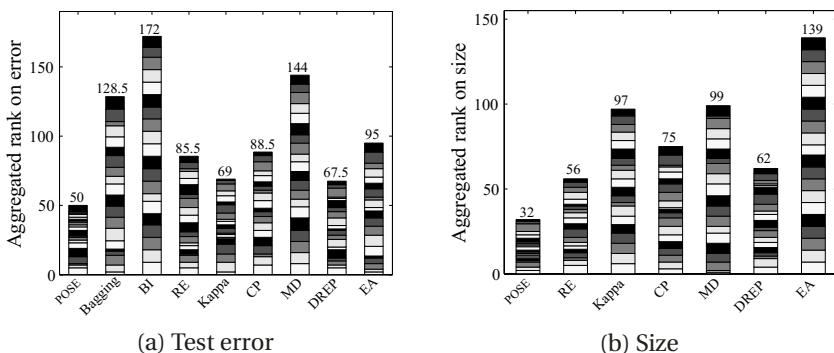


Figure 13.2. The aggregated rank on 20 binary data sets (the smaller, the better).

Table 13.1. The test errors and the sizes (mean \pm std.) of the compared algorithms on 20 binary data sets. In each data set, the smallest values are bolded, and “•/◦” denote respectively that POSE is significantly better/worse than the corresponding algorithm by the t -test with confidence level 0.05. In the rows of the count of the best, the largest values are bolded. The count of direct win denotes the number of data sets on which POSE has a smaller test error/size than the corresponding algorithm (1 tie is counted as 0.5 win), where significant cells by the sign-test with confidence level 0.05 are boxed.

Data set	Test error								
	POSE	Bagging	BI	RE	Kappa	CP	MD	DREP	EA
australian	.144 \pm .020	.143\pm.017	.152 \pm .023•	.144 \pm .020	.143\pm.021	.145 \pm .022	.148 \pm .022	.144 \pm .019	.143\pm.020
breast-cancer	.275\pm.041	.279 \pm .037	.298 \pm .044•	.277 \pm .031	.287 \pm .037	.282 \pm .043	.295 \pm .044•	.275\pm.036	.275\pm.032
disorders	.304\pm.039	.327 \pm .047•	.365 \pm .047•	.320 \pm .044•	.326 \pm .042•	.306 \pm .039	.337 \pm .035•	.316 \pm .045	.317 \pm .046•
heart-statlog	.197 \pm .037	.195 \pm .038	.235 \pm .049•	.187\pm.044	.201 \pm .038	.199 \pm .044	.226 \pm .048•	.194 \pm .044	.196 \pm .032
house-votes	.045 \pm .019	.041\pm.013	.047 \pm .016	.043 \pm .018	.044 \pm .017	.045 \pm .017	.048 \pm .018	.045 \pm .017	.041\pm.012
ionosphere	.088 \pm .021	.092 \pm .025	.117 \pm .022	.086 \pm .021	.084\pm.020	.089 \pm .021	.100 \pm .026•	.085 \pm .021	.093 \pm .026
kr-vs-kp	.010\pm.003	.015 \pm .007•	.011 \pm .004	.010\pm.004	.010\pm.003	.011 \pm .003	.011 \pm .005	.011 \pm .003	.012 \pm .004
letter-ah	.013 \pm .005	.021 \pm .006•	.023 \pm .008•	.015 \pm .006•	.012\pm.006	.015 \pm .006	.017 \pm .007•	.014 \pm .005	.017 \pm .006•
letter-br	.046\pm.008	.059 \pm .013•	.078 \pm .012•	.048 \pm .012	.048 \pm .014	.048 \pm .012	.057 \pm .014•	.048 \pm .009	.053 \pm .011•
letter-oq	.043 \pm .009	.049 \pm .012•	.078 \pm .017•	.046 \pm .011	.042 \pm .011	.042 \pm .010	.046 \pm .011	.041\pm.010	.044 \pm .011
optdigits	.035\pm.006	.038 \pm .007•	.095 \pm .008•	.036 \pm .006	.035\pm.005	.036 \pm .005	.037 \pm .006•	.035\pm.006	.035\pm.006
satimage-12v57	.028\pm.004	.029 \pm .004	.052 \pm .006•	.029 \pm .004	.028\pm.004	.029 \pm .004	.029 \pm .004	.029 \pm .004	.029 \pm .004
satimage-2v5	.021\pm.007	.023 \pm .009	.033 \pm .010	.023 \pm .007	.022 \pm .007	.021\pm.008	.026 \pm .010•	.022 \pm .008	.021\pm.008
sick	.015\pm.003	.018 \pm .004•	.018 \pm .004•	.016 \pm .003	.017 \pm .003•	.016 \pm .003	.017 \pm .003	.016 \pm .003	.017 \pm .004•
sonar	.248\pm.056	.266 \pm .052	.310 \pm .051•	.267 \pm .053•	.249 \pm .059	.250 \pm .048	.268 \pm .055•	.257 \pm .056	.251 \pm .041
spambase	.065\pm.006	.068 \pm .007•	.093 \pm .008•	.066 \pm .006	.066 \pm .006	.066 \pm .006	.068 \pm .007•	.065\pm.006	.066 \pm .006
tic-tac-toe	.131 \pm .027	.164 \pm .028•	.212 \pm .028•	.135 \pm .026	.132 \pm .023	.132 \pm .026	.145 \pm .022•	.129\pm.026	.138 \pm .020
vehicle-bo-vs	.224\pm.023	.228 \pm .026	.257 \pm .025•	.226 \pm .022	.233 \pm .024•	.234 \pm .024•	.244 \pm .024•	.234 \pm .026	.230 \pm .024
vehicle-b-v	.018\pm.011	.027 \pm .014•	.024 \pm .013•	.020 \pm .011	.019 \pm .012	.020 \pm .011	.021 \pm .011•	.019 \pm .013	.026 \pm .013•
vote	.044 \pm .018	.047 \pm .018	.046 \pm .016	.044 \pm .017	.041\pm.016	.043 \pm .016	.045 \pm .014	.043 \pm .019	.045 \pm .015
Count of the best	12	2	0	2	7	1	0	5	5
POSE: Count of direct win	[17]	[20]	[15.5]	12.5	[17]	[20]	12.5	[15.5]	
Ensemble size									
australian	10.6 \pm 4.2	—	—	12.5 \pm 6.0	14.7 \pm 12.6	11.0 \pm 9.7	8.5\pm14.8	11.7 \pm 7.4	41.9 \pm 6.7•
breast-cancer	8.4 \pm 3.5	—	—	8.7 \pm 3.6	26.1 \pm 21.7•	8.8 \pm 12.3	7.8\pm15.2	9.2 \pm 3.7	44.6 \pm 6.6•
disorders	14.7 \pm 4.2	—	—	13.9\pm4.2	24.7 \pm 16.3•	15.3 \pm 10.6	17.7 \pm 20.0	13.9\pm5.9	42.0 \pm 6.2•
heart-statlog	9.3\pm2.3	—	—	11.4 \pm 5.0	17.9 \pm 4.1•	13.2 \pm 8.2•	13.6 \pm 21.1	11.3 \pm 7.2•	44.2 \pm 5.1•
house-votes	2.9\pm1.7	—	—	3.9 \pm 4.0	5.5 \pm 3.3•	4.7 \pm 4.4•	5.9 \pm 14.1	4.1 \pm 2.7•	46.5 \pm 6.1•
ionosphere	5.2\pm2.2	—	—	7.9 \pm 5.7•	10.5 \pm 6.9•	8.5 \pm 6.3•	10.7 \pm 14.6•	8.4 \pm 4.3•	48.8 \pm 5.1•
kr-vs-kp	4.2\pm1.8	—	—	5.8 \pm 4.5	10.6 \pm 9.1•	9.6 \pm 8.6•	7.2 \pm 15.2	7.1 \pm 3.9•	45.9 \pm 5.8•
letter-ah	5.0\pm1.9	—	—	7.3 \pm 4.4•	7.1 \pm 3.8•	8.7 \pm 4.7•	11.0 \pm 10.9•	7.8 \pm 3.6•	42.5 \pm 6.5•
letter-br	10.9\pm2.6	—	—	15.1 \pm 7.3•	13.8 \pm 6.7•	12.9 \pm 6.8	23.2 \pm 17.6•	11.3 \pm 3.5	38.3 \pm 7.8•
letter-oq	12.0 \pm 3.7	—	—	13.6 \pm 5.8	13.9 \pm 6.0	12.3 \pm 4.9	23.0 \pm 15.6•	13.7 \pm 4.9	39.3 \pm 8.2•
optdigits	22.7 \pm 3.1	—	—	25.0 \pm 9.3	25.2 \pm 8.1	21.4\pm7.5	46.8 \pm 23.9	25.0 \pm 8.0	41.4 \pm 7.6•
satimage-12v57	17.1\pm5.0	—	—	20.8 \pm 9.2•	22.1 \pm 10.3•	21.2 \pm 10.0•	37.6 \pm 24.3•	18.1 \pm 4.9	42.7 \pm 5.2•
satimage-2v5	5.7\pm1.7	—	—	6.8 \pm 3.2	7.6 \pm 4.2•	10.9 \pm 7.0•	26.2 \pm 28.1•	7.7 \pm 3.5•	44.1 \pm 4.8•
sick	6.9\pm2.8	—	—	7.5 \pm 3.9	10.9 \pm 6.0•	11.5 \pm 10.0•	8.3 \pm 13.6	11.6 \pm 6.7•	44.7 \pm 8.2•
sonar	11.4 \pm 4.2	—	—	11.0\pm4.1	20.6 \pm 9.3•	13.9 \pm 7.1	20.6 \pm 20.7•	14.4 \pm 5.9•	43.1 \pm 6.4•
spambase	17.5 \pm 4.5	—	—	18.5 \pm 5.0	20.0 \pm 8.1	19.0 \pm 9.9	28.8 \pm 17.0•	16.7\pm4.6	39.7 \pm 6.4•
tic-tac-toe	14.5 \pm 3.8	—	—	16.1 \pm 5.4	17.4 \pm 6.5	15.4 \pm 6.3	28.0 \pm 22.6•	13.6\pm3.4	39.8 \pm 8.2•
vehicle-bo-vs	16.5 \pm 4.5	—	—	15.7 \pm 5.7	16.5 \pm 8.2	11.2\pm5.7	21.6 \pm 20.4	13.2 \pm 5.0•	41.9 \pm 5.6•
vehicle-b-v	2.8\pm1.1	—	—	3.4 \pm 2.1	4.5 \pm 1.6•	5.3 \pm 7.4	2.8\pm3.8	4.0 \pm 3.9	48.0 \pm 5.6•
vote	2.7\pm1.1	—	—	3.2 \pm 2.7	5.1 \pm 2.6•	5.4 \pm 5.2•	6.0 \pm 9.8	3.9 \pm 2.5•	47.8 \pm 6.1•
Count of the best	12	—	—	2	0	2	3	3	0
POSE: Count of direct win	—	—	—	[17]	[19.5]	[18]	[17.5]	[16]	[20]

All the criteria agree that BI is the worst on test error, coinciding with the fact that ensemble is usually better than a single classifier. Compared with RE, which greedily minimizes the validation error, POSE minimizes the validation error and the size simultaneously. POSE achieves significant improvement on the test error as well as the size. This observation supports the theoretical analysis that POSE is more powerful than OSE. As a type of

SOSE, EA produces ensembles with large sizes, which has been observed in previous studies [Zhou et al., 2002, Li and Zhou, 2009]. This also confirms our theoretical result that POSE/OSE can be better than SOSE. Kappa, CP and MD are also OSE algorithms but optimizing diversity-like objectives. These algorithms leave the validation error alone. We find that the individual classifiers have similar performance, as shown by the fact that the average coefficient of variation, i.e., the ratio of the standard deviation to the mean, for the validation errors of 100 individual classifiers is only 0.203, making that optimizing the diversity may work alone. DREP is an OSE algorithm optimizing a combined error and diversity objective, which is shown to be better than the OSE algorithms optimizing only the diversity-like objectives, in both test error and ensemble size from Figure 13.2.

Figure 13.3 studies the influence of the original Bagging size n . It can be observed that POSE always has the smallest error and size, and the ranking order of the algorithms is consistent with Figure 13.2.

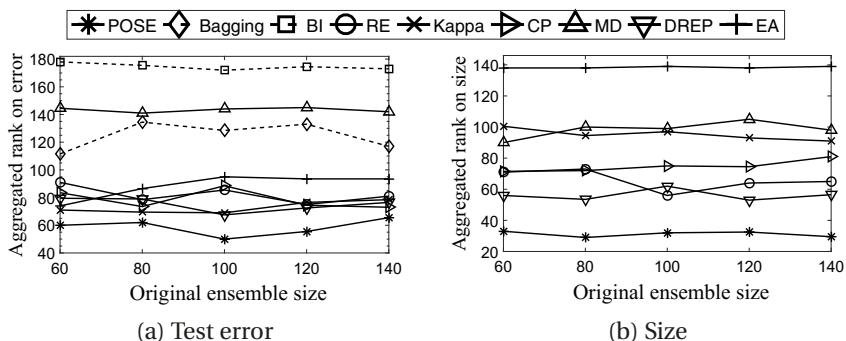


Figure 13.3. The aggregated rank of each algorithm pruning the Bagging of $\{60, \dots, 140\}$ individual classifiers on 20 binary data sets (the smaller, the better).

13.4.2 Multiclass Classification

Next we compare these algorithms on 10 multiclass UCI data sets. Note that Kappa, CP, MD and DREP were originally designed for binary classification, while we extend them for multiclass classification by generalizing their “equal” and “unequal” tests on multiple classes.

The detailed results are shown in Table 13.2. The overall performance is shown in Figure 13.4. It can be observed that the compared algorithms have a similar performance rank as in binary classification except for DREP. DREP performs much worse in multiclass classification than in binary classification, which may be because it was designed and proved only for binary classification scenario [Li et al., 2012].

Table 13.2. The test errors and the sizes (mean \pm std.) of the compared algorithms on 10 multiclass data sets. In each data set, the smallest values are bolded, and “•/◦” denote respectively that POSE is significantly better/worse than the corresponding algorithm by the t -test with confidence level 0.05. In the rows of the count of the best, the largest values are bolded. The count of direct win denotes the number of data sets on which POSE has a smaller test error/size than the corresponding algorithm (1 tie is counted as 0.5 win), where significant cells by the sign-test with confidence level 0.05 are boxed.

Data set	Test error								
	POSE	Bagging	BI	RE	Kappa	CP	MD	DREP	EA
anneal	.017 \pm .006	.032 \pm .013•	.020 \pm .008•	.018 \pm .006	.018 \pm .006	.017\pm.007	.027 \pm .010•	.020 \pm .008•	.025 \pm .010•
audiology	.360\pm.036	.403 \pm .044•	.403 \pm .043•	.365 \pm .040	.370 \pm .035•	.364 \pm .036	.401 \pm .045•	.385 \pm .037•	.383 \pm .036•
balance-scale	.162 \pm .018	.170 \pm .020•	.240 \pm .027•	.165 \pm .026	.160\pm.018	.165 \pm .021	.174 \pm .023•	.167 \pm .020	.166 \pm .023
glass	.307 \pm .049	.322 \pm .051	.377 \pm .054•	.310 \pm .053	.308 \pm .046	.309 \pm .051	.334 \pm .056•	.331 \pm .048•	.312 \pm .041
lymph	.231 \pm .044	.254 \pm .052•	.264 \pm .035•	.235 \pm .045	.221\pm.040	.227 \pm .039	.255 \pm .052•	.252 \pm .037•	.251 \pm .050•
primary-tumor	.604\pm.031	.618 \pm .039•	.655 \pm .036•	.610 \pm .032	.612 \pm .038	.610 \pm .030	.615 \pm .038•	.622 \pm .035•	.604\pm.039
soybean	.096 \pm .019	.127 \pm .022•	.150 \pm .021•	.100 \pm .019•	.101 \pm .015•	.101 \pm .020	.125 \pm .023•	.120 \pm .025•	.106 \pm .019•
vehicle	.280 \pm .021	.281 \pm .024	.340 \pm .031•	.277 \pm .027	.275\pm.022	.277 \pm .023	.280 \pm .025	.279 \pm .021	.277 \pm .022
vowel	.200\pm.030	.222 \pm .030•	.396 \pm .028•	.203 \pm .028	.203 \pm .028	.205 \pm .027	.224 \pm .030•	.214 \pm .027•	.206 \pm .028•
zoo	.129 \pm .047	.175 \pm .045•	.150 \pm .038•	.132 \pm .042	.125\pm.052	.135 \pm .048	.177 \pm .052•	.144 \pm .038•	.160 \pm .042•
Count of the best	6	0	0	4	1	0	0	0	1
POSE: Count of direct win	[10]	[10]	[9]	6	7.5	[9.5]	[9]	8.5	
Ensemble size									
anneal	3.3 \pm 1.8	—	—	3.0\pm2.5	7.0 \pm 6.0•	4.8 \pm 3.8•	12.0 \pm 12.2•	5.1 \pm 7.3	45.5 \pm 7.0•
audiology	7.8\pm3.0	—	—	11.2 \pm 7.5•	14.9 \pm 11.9•	13.0 \pm 10.9•	25.5 \pm 27.1•	12.0 \pm 14.5	45.3 \pm 4.5•
balance-scale	16.3\pm3.0	—	—	18.3 \pm 6.1	26.2 \pm 7.7•	19.9 \pm 10.5•	48.6 \pm 28.0•	30.2 \pm 17.1•	44.7 \pm 6.7•
glass	11.9\pm3.5	—	—	13.1 \pm 6.1	22.2 \pm 15.6•	13.6 \pm 7.4	27.4 \pm 20.6•	19.4 \pm 17.3•	42.0 \pm 5.2•
lymph	6.9\pm1.9	—	—	7.7 \pm 3.2	9.9 \pm 4.6•	9.2 \pm 5.3•	18.3 \pm 24.5•	7.7 \pm 10.0	46.0 \pm 4.7•
primary-tumor	17.8\pm4.8	—	—	18.4 \pm 8.2	48.5 \pm 21.0•	19.6 \pm 13.2	44.4 \pm 24.8•	25.1 \pm 24.4	41.5 \pm 6.2•
soybean	14.4 \pm 3.4	—	—	14.8 \pm 6.0	12.9\pm4.9	13.6 \pm 6.5	48.5 \pm 30.6•	24.8 \pm 17.1•	42.3 \pm 7.4•
vehicle	20.6 \pm 4.7	—	—	19.5 \pm 5.7	20.6 \pm 9.5	17.8\pm10.1	49.1 \pm 29.9•	38.3 \pm 26.6•	41.4 \pm 6.4•
vowel	24.9\pm4.0	—	—	26.8 \pm 6.5	34.8 \pm 10.5•	25.4 \pm 9.8	61.4 \pm 22.8•	44.6 \pm 21.6•	39.6 \pm 5.1•
zoo	3.5 \pm 1.7	—	—	3.8 \pm 3.8	7.0 \pm 5.6•	7.5 \pm 7.0•	19.6 \pm 22.6•	6.9 \pm 8.2•	47.1 \pm 5.4•
Count of the best	7	—	—	1	1	1	0	0	0
POSE: Count of direct win	—	—	—	8	8.5	8	[10]	[10]	[10]

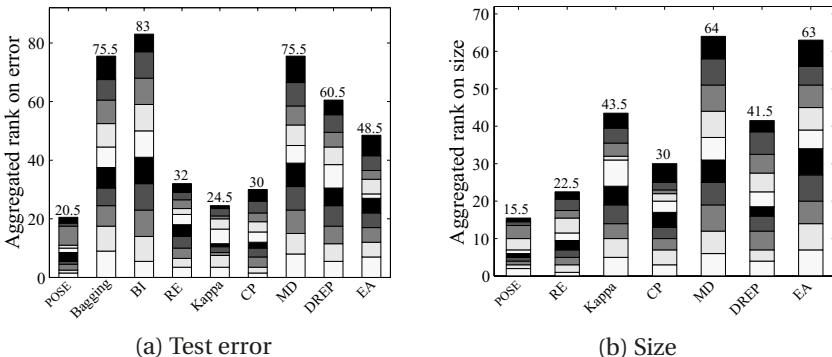


Figure 13.4. The aggregated rank on 10 multiclass data sets (the smaller, the better).

13.4.3 Application

We apply POSE to the task of mobile human activity recognition (MHAR) using smartphones. As smartphones have become more and more popular and crucial in everyday life, human body signals can be retrieved easily

from embedded inertial sensors. Learning from this information can help better monitor user health and understand user behaviors. MHAR using smartphones is to identify the actions carried out by a person according to the context information gathered by smartphones. Besides the accuracy of the classifier, it is important to consider that smartphones have only limited storage and computational resources for doing predictions. Thus, selective ensemble is particularly appealing in the MHAR task.

The MHAR data set [Anguita et al., 2012] is collected from 30 volunteers wearing the smartphone on the waist who performed 6 activities, i.e., walking, upstairs, downstairs, standing, sitting and laying. The embedded 3D-accelerometer and 3D-gyroscope of a Samsung Galaxy S2 smartphone were used to collect data at a constant rate of 50 Hz. The records build a multi-class classification data set with 10,299 instances and 561 attributes. The data set was further randomly partitioned into two parts: 70% for the training set and 30% for the test set. To evaluate the performance of one selective ensemble algorithm on MHAR, we repeat 30 independent runs. In each run, we fix the test set and randomly split the training set into two parts: 75% for training and 25% for validation. Bagging of 100 C4.5 decision trees are firstly trained on the training set, then pruned by the selective ensemble algorithms using the validation set, and finally tested on the test set.

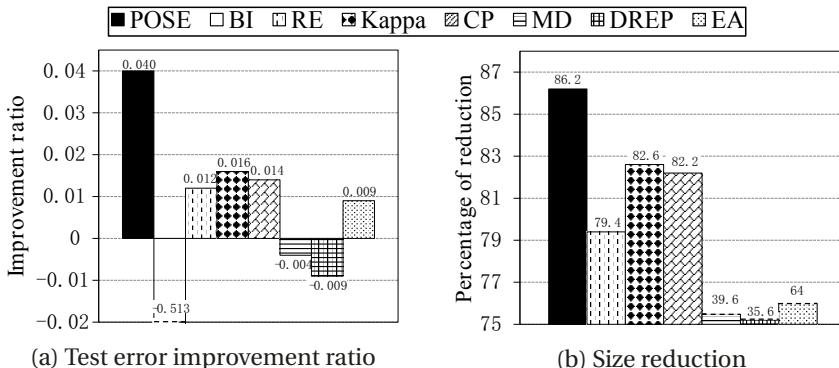


Figure 13.5. Average performance improvement and size reduction from the Bagging of 100 individual classifiers for the MHAR task (the larger, the better).

Figure 13.5(a) depicts the improvement ratio of the selective ensemble algorithms to the test error of the full Bagging, and Figure 13.5(b) shows the reduction percentage of the number of classifiers. POSE achieves the best accuracy, about 3 times more than the runner-up on the improvement ratio. Meanwhile, POSE has the best ensemble size reduction, which saves more than 20%, i.e., $(17.4 - 13.8)/17.4$, storage space than the runner-up. Furthermore, compared with the reported accuracy 89.3% of the multiclass SVM [Anguita et al., 2012], POSE achieves a better one 90.4%.

13.5 Summary

Selective ensemble can further improve the generalization performance of an ensemble, while reducing storage cost and prediction time cost. There are naturally two goals in selective ensemble, i.e., maximizing the generalization ability and minimizing the ensemble size. Most previous selective ensemble methods solve objectives that mix the two goals. In this chapter, we present a Pareto optimization based algorithm, POSE, which solves the explicit bi-objective formulation by an MOEA.

We derive theoretical results revealing the advantage of POSE over the ordering-based selective ensemble methods as well as the heuristic single-objective optimization-based methods. We then conduct experiments, showing the superiority of POSE over the compared state-of-the-art selective ensemble algorithms. Finally, we apply POSE to the task of mobile human activity recognition, where the prediction accuracy gets improved while the cost of storage gets reduced. It can be expected that larger performance gains and cost reduction can be achieved by designing better Pareto optimization based selective ensemble methods.



Subset Selection

In the previous chapter, we present an effective Pareto optimization based algorithm for the selective ensemble problem. This chapter studies a more general problem, i.e., *subset selection*. Subset selection, as illustrated in Figure 14.1, is to select a subset of size at most b from a total set of n items for maximizing (or minimizing) some given objective function f . This problem arises in various real applications, such as *maximum coverage* [Feige, 1998], *influence maximization* [Kempe et al., 2003], *sensor placement* [Sharma et al., 2015], to name a few. Particularly, many important machine learning tasks, such as *feature selection* [Liu and Motoda, 1998] and *sparse regression* [Miller, 2002], are all closely related to subset selection. The subset selection problem is, however, generally NP-hard [Natarajan, 1995, Davis et al., 1997]. The greedy algorithm, which iteratively selects one item with the largest marginal gain, has been shown to be a good approximation solver. When the involved objective functions satisfy the *monotone* and *submodular* property, which will be introduced in Section 14.1, the greedy algorithm can achieve the $(1 - 1/e)$ -approximation guarantee [Nemhauser et al., 1978], which is optimal in general [Nemhauser and Wolsey, 1978]. For the application of sparse regression, where f can be non-submodular, the greedy algorithm can achieve the best-so-far approximation guarantee of $1 - e^{-\gamma}$, where γ is the submodularity ratio measuring how close f is to submodularity [Das and Kempe, 2011].

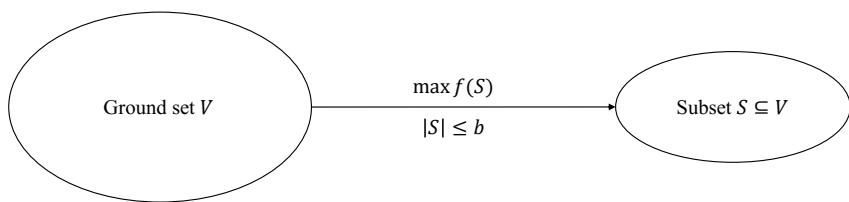


Figure 14.1. Illustration of the subset selection problem.

In this chapter, we present the POSS algorithm [Qian et al., 2015c], which treats subset selection as a bi-objective optimization problem maximizing some given objective function and minimizing the subset size simultaneously, then solves the problem by an MOEA, and finally selects the best feasible solution from the generated non-dominated solution set. Note that the subset selection problem also covers the selective ensemble problem studied in the previous chapter. In this chapter, however, we explicitly consider the monotone and submodular properties of the objective function; in other words, we further exploit the inner problem structure endowed by the objective function. We prove that POSS using polynomial time obtains an approximation guarantee of $(1 - e^{-\gamma})$ for monotone objective functions. When the objective function is submodular, $\gamma = 1$ and then the approximation guarantee becomes $1 - 1/e$, which is optimal in general. For sparse regression, the approximation guarantee $(1 - e^{-\gamma})$ by POSS also matches the best-known one, previously obtained by the greedy algorithm [Das and Kempe, 2011]. We also theoretically compare POSS with the greedy algorithm on the *Exponential Decay* subclass of sparse regression, which has clear applications in sensor networks [Das and Kempe, 2008]. We prove that POSS can efficiently find an optimal solution, whereas the greedy algorithm cannot. The experimental results on sparse regression exhibit the excellent performance of POSS.

The rest of this chapter is organized as follows. Section 14.1 introduces the subset selection problem and its appearance in various real applications. Sections 14.2 to 14.4 present the POSS algorithm, theoretical analysis and empirical study, respectively. Section 14.5 concludes this chapter.

14.1 Subset Selection

In this section, we first introduce the general subset selection problem, and then give some concrete examples.

Given a ground set $V = \{v_1, v_2, \dots, v_n\}$, we study the functions $f : 2^V \rightarrow \mathbb{R}$ over subsets of V . A set function f is monotone if $\forall S \subseteq T : f(S) \leq f(T)$. Without loss of generality, we assume that monotone functions are normalized, i.e., $f(\emptyset) = 0$. A set function $f : 2^V \rightarrow \mathbb{R}$ is submodular [Nemhauser et al., 1978] iff $\forall S, T \subseteq V$:

$$f(S \cup T) + f(S \cap T) \leq f(S) + f(T), \quad (14.1)$$

or equivalently, $\forall S \subseteq T \subseteq V$:

$$f(T) - f(S) \leq \sum_{v \in T \setminus S} (f(S \cup v) - f(S)), \quad (14.2)$$

or equivalently, $\forall S \subseteq T \subseteq V, v \notin T$:

$$f(S \cup v) - f(S) \geq f(T \cup v) - f(T). \quad (14.3)$$

Intuitively, Eq. (14.3) means that a submodular function satisfies the diminishing returns property, i.e., the incremental value of the function made by adding a single item to an input set decreases as the input set extends. Note that a singleton set $\{v\}$ is represented by v for simplicity.

For a general set function f , the submodularity ratio in Definition 14.1 characterizes how close f is to submodularity. For a monotone function f , it holds that $0 \leq \gamma_{U,k}(f) \leq 1$, and f is submodular iff $\forall U, k : \gamma_{U,k}(f) = 1$. For some concrete non-submodular functions, bounds of $\gamma_{U,k}(f)$ have been derived [Das and Kempe, 2011, Bian et al., 2017, Elenberg et al., 2018]. When f is clear, we will use $\gamma_{U,k}$ for short.

Definition 14.1 (Submodularity Ratio [Das and Kempe, 2011]). Let f be a non-negative set function. The submodularity ratio of f with respect to a set U and a parameter $k \geq 1$ is

$$\gamma_{U,k}(f) = \min_{L \subseteq U, S: |S| \leq k, S \cap L = \emptyset} \frac{\sum_{v \in S} (f(L \cup v) - f(L))}{f(L \cup S) - f(L)}. \quad (14.4)$$

Given a ground set $V = \{v_1, v_2, \dots, v_n\}$, an objective function $f : 2^V \rightarrow \mathbb{R}$ and a budget $b \in [n]$, the goal is to select a subset $S \subseteq V$ such that f is maximized with the constraint $|S| \leq b$. A subset of V can be naturally represented by a binary vector $s \in \{0, 1\}^n$, where $s_i = 1$ if the i -th item of V is selected and $s_i = 0$ otherwise. $s \in \{0, 1\}^n$ and its corresponding subset will not be distinguished for notational convenience. The subset selection problem is formally stated as follows.

Definition 14.2 (Subset Selection). Given a set of items $V = \{v_1, v_2, \dots, v_n\}$, an objective function $f : \{0, 1\}^n \rightarrow \mathbb{R}$ and a budget $b \in [n]$, to find

$$s^* = \arg \max_{s \in \{0, 1\}^n} f(s) \quad \text{s.t.} \quad |s|_1 \leq b. \quad (14.5)$$

The subset selection problem is NP-hard in general [Natarajan, 1995, Davis et al., 1997]. The greedy algorithm has been shown to be a good approximation solver. As presented in Algorithm 14.1, the greedy algorithm iteratively selects an item with the largest improvement on the objective function f , until b items are selected. When the objective function f is monotone submodular, it can achieve the optimal approximation guarantee $(1 - 1/e)$ [Nemhauser et al., 1978, Nemhauser and Wolsey, 1978].

In the following, we introduce five applications of subset selection with monotone objective functions, i.e., maximum coverage, influence maximization, information coverage maximization, sensor placement and sparse regression, which will be studied in this book. The objective functions of the former four applications are submodular, whereas that of the last can be non-submodular. Note that the objective functions of some subset selection applications, such as selective ensemble and feature selection, can even be non-monotone.

Algorithm 14.1 Greedy Algorithm

Input: $V = \{v_1, v_2, \dots, v_n\}$; objective function $f : \{0, 1\}^n \rightarrow \mathbb{R}$; budget $b \in [n]$
Output: solution $s \in \{0, 1\}^n$ with $|s|_1 = b$
Process:

- 1: let $s = 0^n$ and $t = 0$;
 - 2: **while** $t < b$ **do**
 - 3: let \hat{v} be an item maximizing $f(s \cup v)$, i.e., $\hat{v} = \arg \max_{v \notin s} f(s \cup v)$;
 - 4: let $s = s \cup \hat{v}$, and $t = t + 1$
 - 5: **end while**
 - 6: **return** s
-

14.1.1 Maximum Coverage

Maximum coverage in Definition 14.3 is a classic combinatorial optimization problem [Feige, 1998]. Given a family of sets that cover a universe of elements, the goal is to select at most b sets whose union is maximum. The objective function $|\cup_{i:s_i=1} S_i|$ is monotone and submodular.

Definition 14.3 (Maximum Coverage). *Given a ground set U , a collection $V = \{S_1, S_2, \dots, S_n\}$ of subsets of U , and a budget b , to find*

$$s^* = \arg \max_{s \in \{0,1\}^n} |\cup_{i:s_i=1} S_i| \quad \text{s.t.} \quad |s|_1 \leq b. \quad (14.6)$$

14.1.2 Influence Maximization

Influence maximization is to identify a set of influential users in social networks. Let a directed graph $G = (V, E)$ represent a social network, where each node is a user and each edge $(u, v) \in E$ has a probability $p_{u,v}$ representing the strength of influence from user u to v . Given a budget b , influence maximization in Definition 14.4 is to find a subset S of V with $|S| \leq b$ such that the expected number of nodes activated by propagating from S , called influence spread, is maximized [Kempe et al., 2003]. The fundamental propagation model Independence Cascade (IC) is employed here. As shown in Definition 14.5, it uses a set A_t to record the nodes activated at time t , and at time $t + 1$, each inactive neighbor v of $u \in A_t$ becomes active with probability $p_{u,v}$. This process is repeated until no nodes get activated at some time. The set of nodes activated by propagating from S is denoted as $A(S)$, which is a random variable. Thus, $\mathbb{E}[|A(S)|]$ represents the expected number of active nodes, which has been proved to be monotone and submodular [Kempe et al., 2003].

Definition 14.4 (Influence Maximization). *Given a directed graph $G = (V, E)$ with $|V| = n$, edge probabilities $p_{u,v}$ where $(u, v) \in E$, and a budget b , to find*

$$s^* = \arg \max_{s \in \{0,1\}^n} \mathbb{E}[|A(s)|] \quad \text{s.t.} \quad |s|_1 \leq b. \quad (14.7)$$

Definition 14.5 (IC). Given a directed graph $G = (V, E)$ with edge probabilities $p_{u,v}$ where $(u, v) \in E$, and a seed set $S \subset V$, it propagates as follows:

1. let $A_0 = S$ and $t = 0$;
2. repeat until $A_t = \emptyset$
3. for each edge (u, v) with $u \in A_t$ and $v \in V \setminus \cup_{i \leq t} A_i$
4. v is added into A_{t+1} with probability $p_{u,v}$
5. let $t = t + 1$

14.1.3 Information Coverage Maximization

Considering that an inactive node may be informed of information by any of its active neighbor nodes, Wang et al. [2015] formalized a new problem, i.e., information coverage maximization, to maximize the expected number of both active nodes and informed nodes. An inactive node is informed if there exists at least one active neighbor node. For each $v \in A(S)$, let $N(v)$ denote the set of inactive neighbor nodes of v . Then, the set of active nodes and informed nodes by propagating from S can be represented as

$$AI(S) = A(S) \cup (\cup_{v \in A(S)} N(v)). \quad (14.8)$$

The objective function $\mathbb{E}[|AI(S)|]$ has been proved to be monotone and submodular [Wang et al., 2015].

Definition 14.6 (Information Coverage Maximization). Given a directed graph $G = (V, E)$ with $|V| = n$, edge probabilities $p_{u,v}$ where $(u, v) \in E$, and a budget b , to find

$$\mathbf{s}^* = \arg \max_{\mathbf{s} \in \{0,1\}^n} \mathbb{E}[|AI(\mathbf{s})|] \quad s.t. \quad |\mathbf{s}|_1 \leq b. \quad (14.9)$$

14.1.4 Sensor Placement

Given a limited number of sensors, the sensor placement problem is to decide where to place them such that the uncertainty is mostly reduced. Let $\mathbf{s} \in \{0,1\}^n$ represent a placement of sensors on n locations $\{v_1, v_2, \dots, v_n\}$, where $s_j = 1$ means that location v_j is installed with a sensor. Let O_j denote a random variable representing the observations collected from location v_j by installing a sensor. Note that the conditional entropy, i.e., remaining uncertainty, of a total set U of random variables given a subset S is $H(U | S) = H(U) - H(S)$, where $H(\cdot)$ denotes the entropy. Thus, minimizing the uncertainty of U is equivalent to maximizing the entropy of S . Let $U = \{O_j \mid j \in [n]\}$. Then, sensor placement is to maximize the entropy of $\{O_j \mid s_j > 0\}$ using at most b sensors [Krause et al., 2008b], as shown in Definition 14.7. Note that the entropy $H(\cdot)$ is monotone and submodular. Sharma et al. [2015] have shown that the greedy algorithm actually can achieve a nearly optimal solution by proving that the submodular objective function is close to modular.

Definition 14.7 (Sensor Placement). Given n locations $V = \{v_1, v_2, \dots, v_n\}$ and a budget b , to find

$$\mathbf{s}^* = \arg \max_{\mathbf{s} \in \{0,1\}^n} H(\{O_j \mid s_j > 0\}) \quad \text{s.t.} \quad |\mathbf{s}|_1 \leq b, \quad (14.10)$$

where O_j denotes a random variable representing the observations collected from location v_j by installing a sensor.

14.1.5 Sparse Regression

Sparse regression [Miller, 2002] is to find a sparse approximation solution to the linear regression problem, where the solution vector can have only a few non-zero elements. Without loss of generality, assume that all random variables are normalized to have expectation 0 and variance 1.

Definition 14.8 (Sparse Regression). Given all observation variables $V = \{v_1, v_2, \dots, v_n\}$, a predictor variable z and a budget b , to find a set of at most b variables minimizing the mean squared error, i.e.,

$$\arg \min_{\mathbf{s} \in \{0,1\}^n} \text{MSE}_{z,\mathbf{s}} \quad \text{s.t.} \quad |\mathbf{s}|_1 \leq b, \quad (14.11)$$

where the mean squared error of a subset $\mathbf{s} \in \{0, 1\}^n$ is defined as

$$\text{MSE}_{z,\mathbf{s}} = \min_{\boldsymbol{\alpha} \in \mathbb{R}^{|\mathbf{s}|_1}} \mathbb{E} \left[\left(z - \sum_{i:s_i=1} \alpha_i v_i \right)^2 \right]. \quad (14.12)$$

For the ease of theoretical treatment, the *squared multiple correlation*

$$R_{z,\mathbf{s}}^2 = \frac{\text{Var}(z) - \text{MSE}_{z,\mathbf{s}}}{\text{Var}(z)} \quad (14.13)$$

is used to replace $\text{MSE}_{z,\mathbf{s}}$ [Diekhoff, 1992, Johnson and Wichern, 2007], and thus, the sparse regression problem is equivalently

$$\arg \max_{\mathbf{s} \in \{0,1\}^n} R_{z,\mathbf{s}}^2 \quad \text{s.t.} \quad |\mathbf{s}|_1 \leq b. \quad (14.14)$$

Note that the objective function $R_{z,\mathbf{s}}^2$ is monotone but not necessarily submodular. As all random variables are normalized to have expectation 0 and variance 1, $R_{z,\mathbf{s}}^2$ is simplified to be $1 - \text{MSE}_{z,\mathbf{s}}$. Das and Kempe [2011] proved that the forward regression (FR) algorithm can produce a solution S^{FR} with $|S^{FR}| = b$ and $R_{z,S^{FR}}^2 \geq (1 - e^{-\gamma_{S^{FR},b}}) \cdot \text{OPT}$, which is the best currently known approximation guarantee. The FR algorithm is just the greedy algorithm with f specialized as $R_{z,\mathbf{s}}^2$.

14.2 The POSS Algorithm

In this section, we present the Pareto Optimization algorithm for Subset Selection (POSS). The subset selection problem in Eq. (14.5) can be separated into two objectives, one optimizes the original objective function, i.e., $\max_{\mathbf{s} \in \{0,1\}^n} f(\mathbf{s})$, meanwhile the other keeps the subset size small, i.e., $\min_{\mathbf{s} \in \{0,1\}^n} \max\{|\mathbf{s}|_1 - b, 0\}$. Usually the two objectives are conflicting, as a subset with a better f value can have a larger size. POSS solves these two objectives simultaneously, described as follows.

The original problem Eq. (14.5) is reformulated as a bi-objective maximization problem

$$\arg \max_{\mathbf{s} \in \{0,1\}^n} (f_1(\mathbf{s}), f_2(\mathbf{s})), \quad (14.15)$$

where

$$f_1(\mathbf{s}) = \begin{cases} -\infty, & |\mathbf{s}|_1 \geq 2b \\ f(\mathbf{s}), & \text{otherwise} \end{cases}, \quad f_2(\mathbf{s}) = -|\mathbf{s}|_1. \quad (14.16)$$

Note that setting f_1 to $-\infty$ is to exclude overly infeasible solutions, i.e., solutions with size at least $2b$. Inspired by the SEIP framework presented in Chapter 6, we further introduce the isolation function $I : \{0,1\}^n \rightarrow \mathbb{R}$, which determines if two solutions are allowed to be compared: they are comparable only if they have the same isolation function value. The implementation of I is left as a parameter of the algorithm, whereas its influence will be clear in the analysis. Note that when I is set to a constant function, it can be ignored as every solution has the same I value.

To compare two solutions \mathbf{s} and \mathbf{s}' , we first judge if they have the same isolation function value. If not, we say that they are *incomparable*. If they have the same isolation function value, the domination relationship in Definition 1.2 can be used to compare them. Thus, \mathbf{s} is *worse* than \mathbf{s}' if $I(\mathbf{s}) = I(\mathbf{s}')$ and $\mathbf{s} \preceq \mathbf{s}'$; \mathbf{s} is *strictly worse* than \mathbf{s}' if $I(\mathbf{s}) = I(\mathbf{s}')$ and $\mathbf{s} \prec \mathbf{s}'$; otherwise, they are incomparable.

After transformation, GSEMO with a special initial solution 0^n , i.e., the empty set, can be employed to solve the bi-objective maximization problem. The procedure of POSS is described in Algorithm 14.2. Starting from the population P containing only the solution 0^n , POSS generates new solutions by randomly flipping bits of an archived solution in lines 4 and 5. Newly generated solutions are compared with the previously archived solutions in line 6. If the newly generated solution is not strictly worse than any previously archived solution, it will be added into the population, and meanwhile those archived solutions worse than the newly generated solution will be removed from P .

The iteration of POSS repeats for T times. Note that T is a parameter, which may depend on the available resource of the user. We will theoretically analyze the relationship between the solution quality and T , and will

Algorithm 14.2 POSS Algorithm

Input: $V = \{v_1, v_2, \dots, v_n\}$; objective function $f : \{0, 1\}^n \rightarrow \mathbb{R}$; budget $b \in [n]$
Parameter: number T of iterations; isolation function $I : \{0, 1\}^n \rightarrow \mathbb{R}$
Output: solution $\mathbf{s} \in \{0, 1\}^n$ with $|\mathbf{s}|_1 \leq b$
Process:

```

1: let  $\mathbf{s} = 0^n$  and  $P = \{\mathbf{s}\}$ ;
2: let  $t = 0$ ;
3: while  $t < T$  do
4:   select a solution  $\mathbf{s}$  from  $P$  uniformly at random;
5:   apply bit-wise mutation on  $\mathbf{s}$  to generate  $\mathbf{s}'$ ;
6:   if  $\nexists \mathbf{z} \in P$  such that  $I(\mathbf{z}) = I(\mathbf{s}')$  and  $\mathbf{z} \succ \mathbf{s}'$  then
7:      $Q = \{\mathbf{z} \in P \mid I(\mathbf{z}) = I(\mathbf{s}') \wedge \mathbf{s}' \succeq \mathbf{z}\}$ ;
8:      $P = (P \setminus Q) \cup \{\mathbf{s}'\}$ 
9:   end if
10:   $t = t + 1$ 
11: end while
12: return  $\arg \max_{\mathbf{s} \in P, |\mathbf{s}|_1 \leq b} f_1(\mathbf{s})$ 
```

use the theoretically derived T value in the experiments. After the iterations, we select the final solution from the population in line 12 according to Eq. (14.5); that is, we select the solution with the largest f value, i.e., the largest f_1 value, while the constraint on the set size is kept.

14.3 Theoretical Analysis

In this section, we examine the theoretical property of POSS. We first prove its general approximation guarantee, which reaches the best-known one previously obtained by the greedy algorithm. We then compare it with the greedy algorithm on the application of sparse regression, and show that POSS can be strictly better.

14.3.1 The General Problem

Theorem 14.1 gives a polynomial-time approximation bound of POSS for subset selection with monotone objective functions. Let $\mathbb{E}[T]$ denote the average number T of iterations. The proof relies on the property of f in Lemma 14.1; that is, $\forall \mathbf{s} \in \{0, 1\}^n$, there always exists one item, the inclusion of which can bring an improvement on f proportional to the current distance to the optimal function value.

Lemma 14.1. $\forall \mathbf{s} \in \{0, 1\}^n$, there exists one item $v \notin \mathbf{s}$ such that

$$f(\mathbf{s} \cup v) - f(\mathbf{s}) \geq \frac{\gamma_{\mathbf{s}, b}}{b} (\text{OPT} - f(\mathbf{s})). \quad (14.17)$$

Proof. Let s^* be an optimal solution, i.e., $f(s^*) = \text{OPT}$. From the definition of submodularity ratio in Definition 14.1, we have

$$\sum_{v \in s^* \setminus s} (f(s \cup v) - f(s)) \geq \gamma_{s,b} \cdot (f(s \cup s^*) - f(s)). \quad (14.18)$$

Note that $|s^* \setminus s|_1 \leq b$ and $f(s \cup s^*) \geq f(s^*) = \text{OPT}$ by monotonicity. Let $\hat{v} = \arg \max_{v \in s^* \setminus s} f(s \cup v)$. Then, we have

$$f(s \cup \hat{v}) - f(s) \geq \frac{\gamma_{s,b}}{|s^* \setminus s|_1} (f(s \cup s^*) - f(s)) \geq \frac{\gamma_{s,b}}{b} (\text{OPT} - f(s)). \quad (14.19)$$

□

Theorem 14.1. *For subset selection with monotone objective functions, POSS with $\mathbb{E}[T] \leq 2eb^2n$ and $I(\cdot) = 0$, i.e., a constant function, finds a solution s with $|s|_1 \leq b$ and $f(s) \geq (1 - e^{-\gamma_{\min}}) \cdot \text{OPT}$, where $\gamma_{\min} = \min_{s: |s|_1=b-1} \gamma_{s,b}$.*

Proof. Because the isolation function I is a constant function, all solutions are allowed to be compared and it can be ignored. Let J_{\max} denote the maximum value of $j \in [b]$ such that in the population P , $\exists s$ with $|s|_1 \leq j$ and $f(s) \geq (1 - (1 - \gamma_{\min}/b)^j) \cdot \text{OPT}$; that is,

$$J_{\max} = \max \left\{ j \in [b] \mid \exists s \in P : |s|_1 \leq j \wedge f(s) \geq \left(1 - \left(1 - \frac{\gamma_{\min}}{b}\right)^j\right) \text{OPT} \right\}. \quad (14.20)$$

We then analyze the expected number of iterations until $J_{\max} = b$, implying that $\exists s \in P$ satisfying that $|s|_1 \leq b$ and $f(s) \geq (1 - (1 - \gamma_{\min}/b)^b) \cdot \text{OPT} \geq (1 - e^{-\gamma_{\min}}) \cdot \text{OPT}$.

The initial value of J_{\max} is 0, as POSS starts from 0^n . Assume that currently $J_{\max} = i < b$. Let s be a corresponding solution with the value i , i.e., $|s|_1 \leq i$ and $f(s) \geq (1 - (1 - \gamma_{\min}/b)^i) \cdot \text{OPT}$. J_{\max} cannot decrease because cleaning s from P in lines 7 and 8 of Algorithm 14.2 implies that s is weakly dominated by the newly generated solution s' , which must have a smaller size and a larger f value. By Lemma 14.1, flipping one specific 0-bit of s , i.e., adding a specific item, can generate a new solution s' , satisfying $f(s') - f(s) \geq (\gamma_{s,b}/b)(\text{OPT} - f(s))$. Then, we have

$$\begin{aligned} f(s') &\geq \left(1 - \frac{\gamma_{s,b}}{b}\right) f(s) + \frac{\gamma_{s,b}}{b} \cdot \text{OPT} \\ &\geq \left(1 - \left(1 - \frac{\gamma_{s,b}}{b}\right) \left(1 - \frac{\gamma_{\min}}{b}\right)^i\right) \cdot \text{OPT} \\ &\geq \left(1 - \left(1 - \frac{\gamma_{\min}}{b}\right)^{i+1}\right) \cdot \text{OPT}, \end{aligned} \quad (14.21)$$

where Eq. (14.21) holds by $\gamma_{s,b} \geq \gamma_{\min}$, which can be derived from $|s|_1 < b$ and $\gamma_{s,b}$ decreasing with s . Considering $|s'|_1 = |s|_1 + 1 \leq i + 1$, s' will enter into P ; otherwise, according to line 6 of Algorithm 14.2, s' must be dominated by one solution in P , implying that J_{\max} is larger than i , contradicting with the assumption $J_{\max} = i$. After including s' , $J_{\max} \geq i + 1$. Let P_{\max} denote the largest size of P during the running of POSS. Thus, J_{\max} can increase by at least 1 in one iteration with probability at least $(1/P_{\max}) \cdot (1/n)(1 - 1/n)^{n-1} \geq 1/(enP_{\max})$, where $1/P_{\max}$ is a lower bound of the probability of selecting s in line 4 of Algorithm 14.2 and $(1/n)(1 - 1/n)^{n-1}$ is the probability of flipping a specific bit of s while keeping the other bits unchanged in line 5. It requires at most enP_{\max} expected number of iterations to increase J_{\max} . Thus, after $b \cdot enP_{\max}$ iterations in expectation, J_{\max} must have reached b .

By the procedure of POSS, the solutions maintained in P must be incomparable. Thus, each value of one objective can correspond to at most one solution in P . Because the solutions with $|s|_1 \geq 2b$ have $-\infty$ value on the first objective, they must be excluded from P . Thus, $|s|_1 \in \{0, 1, \dots, 2b-1\}$, implying $P_{\max} \leq 2b$. Hence, the expected number of iterations $\mathbb{E}[T]$ for finding the desired solution is at most $2eb^2n$. \square

This approximation bound achieved by POSS reaches the best known guarantee in these two cases: f being any submodular function; the sparse regression problem where f is a specific non-submodular function. When f is submodular, we have $\forall s, b : \gamma_{s,b} = 1$. Thus, the approximation bound in Theorem 14.1 becomes $f(s) \geq (1 - 1/e) \cdot \text{OPT}$, which is optimal in general [Nemhauser and Wolsey, 1978], and is optimal even for the maximum coverage instance, unless $P = NP$ [Feige, 1998]. This best known guarantee was previously achieved by the greedy algorithm [Nemhauser et al., 1978]. For sparse regression, Das and Kempe [2011] proved that the FR algorithm, i.e., the greedy algorithm with f specialized as $R_{z,s}^2$, can produce a solution S^{FR} with $|S^{FR}| = b$ and $R_{z,S^{FR}}^2 \geq (1 - e^{-\gamma_{S^{FR},b}}) \cdot \text{OPT}$, which is the currently best known approximation guarantee. Thus, Theorem 14.1 shows that POSS achieves nearly the best known approximation bound.

14.3.2 Sparse Regression

Next we prove that POSS can be strictly better than the greedy algorithm. We consider a subclass of sparse regression with clear applications in sensor networks [Das and Kempe, 2008], called Exponential Decay as presented in Definition 14.9. In this subclass, the observation variables can be ordered in a line such that their covariances are decreasing exponentially with the distance. Some notations are needed for the analysis. Let $Cov(\cdot, \cdot)$ be the covariance between two random variables, \mathbf{C} be the covariance matrix between all observation variables, i.e., $C_{i,j} = Cov(v_i, v_j)$, and \mathbf{c} be the covariance vector between z and observation variables, i.e., $c_i = Cov(z, v_i)$.

Let \mathbf{C}_S denote the submatrix of \mathbf{C} with row and column set S , and \mathbf{c}_S denote the subvector of \mathbf{c} , containing elements c_i with $v_i \in S$.

Definition 14.9 (Exponential Decay [Das and Kempe, 2008]). *The variables v_i are associated with points $y_1 \leq y_2 \leq \dots \leq y_n$, and $C_{i,j} = a^{|y_i - y_j|}$ for some constant $a \in (0, 1)$.*

In Section 14.3.1, POSS with a constant isolation function has been shown to be able to achieve a generally good approximation guarantee. We prove in Theorem 14.2 that POSS with a proper isolation function can even find an optimal solution on the Exponential Decay subclass. The isolation function $I(\mathbf{s} \in \{0, 1\}^n) = \min\{i \mid s_i = 1\}$ implies that two solutions can be compared only if they have the same minimum index for bit 1. The proof of Theorem 14.2 utilizes the dynamic programming property of the problem in Lemma 14.2.

Lemma 14.2. *[Das and Kempe, 2008] Let $R^2(i, j)$ denote the maximum $R_{z,S}^2$ value by choosing i observation variables, including necessarily v_j , from v_j, \dots, v_n ; that is, $R^2(i, j) = \max\{R_{z,S}^2 \mid S \subseteq \{v_j, \dots, v_n\}, v_j \in S, |S| = i\}$. The following recursive relation holds:*

$$\begin{aligned} R^2(i+1, j) &= \max_{j+1 \leq k \leq n} \left(R^2(i, k) + c_j^2 + (c_j - c_k)^2 \frac{a^{2|y_k - y_j|}}{1 - a^{2|y_k - y_j|}} - 2c_j c_k \frac{a^{|y_k - y_j|}}{1 + a^{|y_k - y_j|}} \right), \end{aligned} \quad (14.22)$$

where the term in big brackets is the R^2 value by adding v_j into the variable subset corresponding to $R^2(i, k)$.

Theorem 14.2. *For the Exponential Decay subclass of sparse regression, POSS with $\mathbb{E}[T] = O(b^2(n-b)n \log n)$ and $I(\mathbf{s} \in \{0, 1\}^n) = \min\{i \mid s_i = 1\}$ finds an optimal solution.*

Proof. We divide the optimization process into $(b+1)$ phases, where the i -th ($1 \leq i \leq b$) phase starts after the $(i-1)$ -th phase finishes. We define that the i -th phase finishes when for each solution corresponding to $R^2(i, j)$ where $1 \leq j \leq n-i+1$, there exists one solution in the population P which weakly dominates it. Let τ_i denote the number of iterations since the $(i-1)$ -th phase has finished, until the i -th phase is completed.

Starting from the solution 0^n , the 0-th phase has finished. We consider τ_i with $i \geq 1$. In this phase, from Lemma 14.2, a solution weakly dominating a corresponding solution of $R^2(i, j)$ can be generated by selecting a specific one from the solutions weakly dominating $R^2(i-1, j+1), \dots, R^2(i-1, n)$ and flipping its j -th bit, occurring with probability at least $(1/P_{\max}) \cdot (1/n)(1-1/n)^{n-1} \geq 1/(enP_{\max})$. Thus, if we have found l desired solutions in the i -th phase, the probability of finding a new desired solution in the next iteration is at least $(n-i+1-l) \cdot (1/enP_{\max})$,

where $(n - i + 1)$ is the total number of desired solutions to find in the i -th phase. Then, $\mathbb{E}[\tau_i] \leq \sum_{l=0}^{n-i} enP_{\max}/(n - i + 1 - l) = O(n \log n P_{\max})$. Therefore, the expected number $\mathbb{E}[T]$ of iterations is $O(bn \log n P_{\max})$ until the b -th phase is finished, implying that an optimal solution corresponding to $\max_{1 \leq j \leq n} R^2(b, j)$ has been found. Note that $P_{\max} \leq 2b(n - b)$, because the incomparable property of the maintained solutions by POSS ensures that there exists at most one solution in P for each possible combination of $|s|_1 \in \{0, 1, \dots, 2b - 1\}$ and $I(s) \in \{0, 1, \dots, n\}$. Thus, $\mathbb{E}[T]$ for finding an optimal solution is $O(b^2(n - b)n \log n)$. \square

We next prove in Theorem 14.3 that the greedy algorithm is blocked from finding an optimal solution on a simple example of the Exponential Decay subclass, as presented in Example 14.1. It can be verified that $\text{Cov}(v_i, v_j) = \prod_{k=i+1}^j d_k$ for $i < j$, and Example 14.1 belongs to the Exponential Decay subclass by letting $y_1 = 0$ and $y_i = \sum_{k=2}^i \log_a d_k$ for $i \geq 2$.

Example 14.1. $v_1 = u_1$, and $\forall i \geq 2 : v_i = d_i v_{i-1} + u_i$, where $d_i \in (0, 1)$, and u_i are independent random variables with expectation 0 such that each v_i has variance 1.

Theorem 14.3. *For Example 14.1 where $n = 3$, $\text{Cov}(u_1, z) = \text{Cov}(u_2, z) = \delta$, $\text{Cov}(u_3, z) = 0.505\delta$, $d_2 = 0.03$ and $d_3 = 0.5$, the greedy algorithm cannot find the optimal solution for $b = 2$.*

Proof. The covariances between v_i and z are $c_1 = \delta$, $c_2 = 0.03c_1 + \delta = 1.03\delta$ and $c_3 = 0.5c_2 + 0.505\delta = 1.02\delta$. As v_i and z have expectation 0 and variance 1, $R_{z,S}^2$ can be represented simply as $c_S^T C_S^{-1} c_S$ [Johnson and Wichern, 2007]. We then calculate the R^2 value as follows: $R_{z,v_1}^2 = \delta^2$, $R_{z,v_2}^2 = 1.0609\delta^2$, $R_{z,v_3}^2 = 1.0404\delta^2$, $R_{z,\{v_1,v_2\}}^2 = 2.0009\delta^2$, $R_{z,\{v_1,v_3\}}^2 = 2.0103\delta^2$, $R_{z,\{v_2,v_3\}}^2 = 1.4009\delta^2$. The optimal solution for $b = 2$ is $\{v_1, v_3\}$. The greedy algorithm first selects v_2 as R_{z,v_2}^2 is the largest, and then selects v_1 as $R_{z,\{v_2,v_1\}}^2 > R_{z,\{v_2,v_3\}}^2$; thus produces the solution $\{v_1, v_2\}$. \square

14.4 Empirical Study

In this section, we empirically study the performance of POSS on sparse regression. We conduct experiments on 12 data sets[†] in Table 14.1 to compare POSS with the following algorithms:

- **FR** [Miller, 2002] iteratively adds one variable with the largest improvement on R^2 , which is actually the greedy algorithm.

[†]<http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/> and <http://archive.ics.uci.edu/ml/>. Some binary classification data sets are used for regression. All variables are normalized to have mean 0 and variance 1.

Table 14.1. Data sets and the corresponding number of instances and features.

Data set	#inst	#feat	Data set	#inst	#feat	Data set	#inst	#feat
<i>housing</i>	506	13	<i>sonar</i>	208	60	<i>clean1</i>	476	166
<i>eunite2001</i>	367	16	<i>triazines</i>	186	60	<i>w5a</i>	9,888	300
<i>svmguide3</i>	1,284	21	<i>coil2000</i>	9,000	86	<i>gisette</i>	7,000	5,000
<i>ionosphere</i>	351	34	<i>mushrooms</i>	8,124	112	<i>farm-ads</i>	4,143	54,877

- **OMP** [Tropp, 2004] iteratively adds one variable which mostly correlates with the predictor variable residual.
- **FoBa** [Zhang, 2011] is based on OMP but deletes one variable adaptively when beneficial. Set parameter $\nu = 0.5$, the solution path length is five times as long as the maximum sparsity level, i.e., $5 \times b$, and the last active set containing b variables is used as the final selection [Zhang, 2011].
- **RFE** [Guyon et al., 2002] iteratively deletes one variable with the smallest weight by linear regression.
- **Lasso** [Tibshirani, 1996], **SCAD** [Fan and Li, 2001] and **MCP** [Zhang, 2010] replaces the ℓ_0 norm constraint with the ℓ_1 norm penalty, the smoothly clipped absolute deviation penalty and the minimax concave penalty, respectively. For implementing these algorithms, we use the SparseReg toolbox developed in [Zhou et al., 2012, Zhou, 2013].

For POSS, we use $I(\cdot) = 0$ as it is generally good, and the number T of iterations is set to $\lfloor 2eb^2n \rfloor$ suggested by Theorem 14.1. To evaluate how far these algorithms are from the optimal function value, we also compute the optimal function value by exhaustive enumeration, denoted as **OPT**.

To assess each algorithm on each data set, the following process is repeated for 100 times. The data set is randomly and evenly split into a training set and a test set. Sparse regression is built on the training set, and evaluated on the test set. We report the average training and test R^2 values.

14.4.1 Optimization Performance

Table 14.2 summarizes the training R^2 for $b = 8$. Note that the results of Lasso, SCAD and MCP are very close, and we only report that of MCP. By the t -test with confidence level 0.05, POSS is shown significantly better than all compared algorithms on all data sets.

We plot the performance curves on two data sets for $b \leq 8$ in Figure 14.2. For *sonar*, OPT is calculated only for $b \leq 5$. It can be observed that POSS tightly follows OPT, and has a clear advantage over the rest algorithms. FR, FoBa and OMP have close performance, much better than MCP, SCAD and Lasso. The poor performance of Lasso is consistent with the previous results in [Das and Kempe, 2011, Zhang, 2011]. Note that although the ℓ_1 norm constraint is a tight convex relaxation of the ℓ_0 norm constraint and can have good results in sparse recovery tasks, the performance of Lasso

Table 14.2. The training R^2 value (mean \pm std.) of the compared algorithms on 12 data sets for $b = 8$. In each data set, “•/◦” denote respectively that POSS is significantly better/worse than the corresponding algorithm by the t -test with confidence level 0.05. “-” means that no results were obtained after running 48 hours.

Data set	OPT	POSS	FR	FoBa	OMP	RFE	MCP
<i>housing</i>	.7437±.0297	.7437±.0297	.7429±.0300•	.7423±.0301•	.7415±.0300•	.7388±.0304•	.7354±.0297•
<i>eunite2001</i>	.8484±.0132	.8482±.0132	.8348±.0143•	.8442±.0144•	.8349±.0150•	.8424±.0153•	.8320±.0150•
<i>sumguide3</i>	.2705±.0255	.2701±.0257	.2615±.0260•	.2601±.0279•	.2557±.0270•	.2136±.0325•	.2397±.0237•
<i>ionosphere</i>	.5995±.0326	.5990±.0329	.5920±.0352•	.5929±.0346•	.5921±.0353•	.5832±.0415•	.5740±.0348•
<i>sonar</i>	—	.5365±.0410	.5171±.0440•	.5138±.0432•	.5112±.0425•	.4321±.0636•	.4496±.0482•
<i>triazines</i>	—	.4301±.0603	.4150±.0592•	.4107±.0600•	.4073±.0591•	.3615±.0712•	.3793±.0584•
<i>coil2000</i>	—	.0627±.0076	.0624±.0076•	.0619±.0075•	.0619±.0075•	.0363±.0141•	.0570±.0075•
<i>mushrooms</i>	—	.9912±.0020	.9909±.0021•	.9909±.0022•	.9909±.0022•	.6813±.1294•	.8652±.0474•
<i>clean1</i>	—	.4368±.0300	.4169±.0299•	.4145±.0309•	.4132±.0315•	.1596±.0562•	.3563±.0364•
<i>w5a</i>	—	.3376±.0267	.3319±.0247•	.3341±.0258•	.3313±.0246•	.3342±.0276•	.2694±.0385•
<i>gisette</i>	—	.7265±.0098	.7001±.0116•	.6747±.0145•	.6731±.0134•	.5360±.0318•	.5709±.0123•
<i>farm-ads</i>	—	.4217±.0100	.4196±.0101•	.4170±.0113•	.4170±.0113•	—	.3771±.0110•
POSS: Win/tie/loss	—	12/0/0	12/0/0	12/0/0	11/0/0	12/0/0	

is not as good as POSS and greedy methods on most data sets. This may be because, unlike assumed in sparse recovery tasks, there may not exist a sparse structure in the data sets. In this case, the ℓ_1 norm constraint can be a poor approximation of the ℓ_0 norm constraint. Meanwhile, the ℓ_1 norm constraint also shifts the optimization problem, making it hard to optimize the original R^2 criterion well.

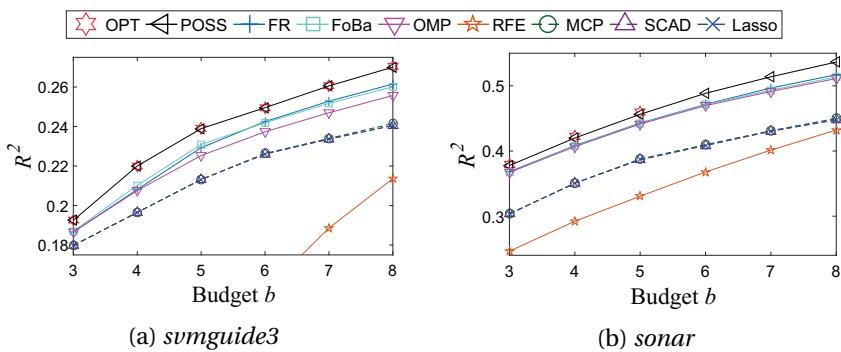


Figure 14.2. Training R^2 (the larger, the better).

14.4.2 Generalization Performance

When testing sparse regression on the test data, it has been known that the sparsity alone may not be a good complexity measure [Zhang, 2011], because it only restricts the number of variables, whereas the range of the variables is unrestricted. Thus, better optimization does not always lead to

better generalization performance. We also observe this in Figure 14.3. On *svmguide3*, test R^2 is consistent with training R^2 in Figure 14.2(a), whereas on *sonar*, better training R^2 in Figure 14.2(b) leads to worse test R^2 in Figure 14.3(b), which may be because the small number of instances makes it prone to overfitting.

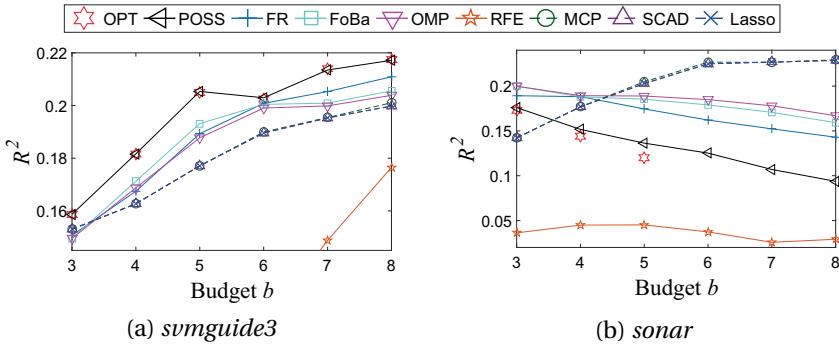


Figure 14.3. Test R^2 (the larger, the better).

As suggested in [Zhang, 2011], other regularization terms may be necessary. We add the ℓ_2 norm regularization into the objective function, i.e.,

$$RSS_{z,s} = \min_{\alpha \in \mathbb{R}^{|s|_1}} \mathbb{E} \left[\left(z - \sum_{i:s_i=1} \alpha_i v_i \right)^2 \right] + \lambda |\alpha|^2. \quad (14.23)$$

The optimization problem is now

$$\arg \min_{s \in \{0,1\}^n} RSS_{z,s} \quad s.t. \quad |s|_1 \leq b. \quad (14.24)$$

We then test all compared algorithms to solve this optimization problem with $\lambda = 0.9615$ on *sonar*. As shown in Figure 14.4, POSS always does the best optimization on the training RSS , and by introducing the ℓ_2 norm, it leads to the best generalization performance in R^2 .

14.4.3 Running Time

Considering the running time in the number of objective function evaluations, OPT does exhaustive search, and thus requires $\binom{n}{b} \geq n^b/b^b$ time, which may be unacceptable for a large data set. FR, FoBa and OMP are greedy-like approaches, and thus are efficient; their running time are all in the order of bn . POSS finds the solutions closest to those of OPT, taking $2eb^2n$ time. Although POSS is slower by a factor of b , the difference would be small when b is a small constant.

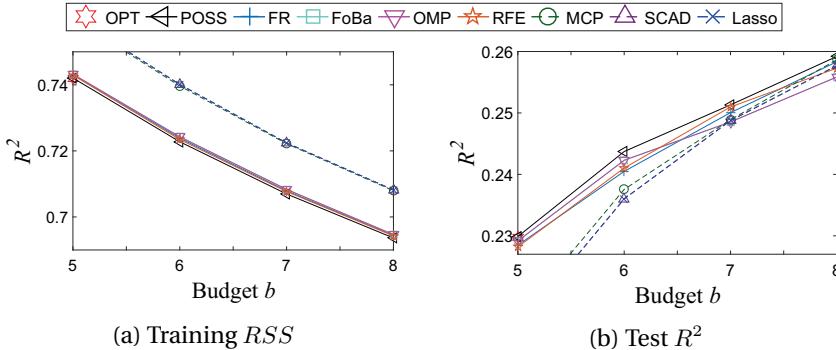


Figure 14.4. Sparse regression with ℓ_2 regularization on *sonar*. RSS : the smaller, the better; R^2 : the larger, the better.

As the $2eb^2n$ time is a theoretical upper bound for POSS to be as good as FR, we empirically examine how tight this bound is. By selecting FR for the baseline, we plot the curve of the R^2 value over the running time for POSS on the two largest data sets *gisette* and *farm-ads*, as shown in Figure 14.5. Here, we do not split the training and test set, and the curve for POSS is the average of 30 independent runs. The x -axis is in bn , the running time of FR. It can be observed that POSS takes about only 14% and 23% of the theoretical time to achieve better performance, respectively, on the two data sets, implying that POSS can be more efficient in practice.

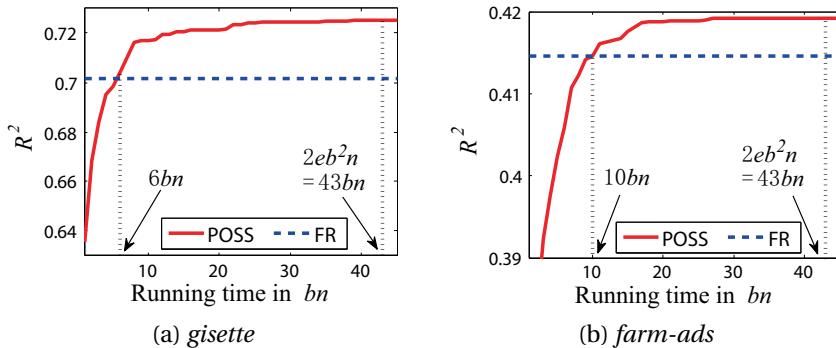


Figure 14.5. Performance vs. running time of POSS and FR.

14.5 Summary

In this chapter, we study the problem of subset selection. The general goal is to select a subset of size b from a large set of n items such that a given

objective function is optimized. Based on Pareto optimization, we present the POSS algorithm that solves the two objectives of the subset selection problem simultaneously, i.e., optimizing the objective function and reducing the subset size.

For subset selection with monotone objective functions, we theoretically prove that POSS with a constant isolation function can generally achieve the best-known approximation guarantee, using time $2eb^2n$. This guarantee was previously obtained by the greedy algorithm. On sparse regression, a representative application of subset selection, we prove that, with a proper isolation function, POSS finds an optimal solution for an important subclass Exponential Decay using time $O(b^2(n - b)n \log n)$, whereas the greedy algorithm may not find an optimal solution. We verify the excellent performance of POSS by experiments, and show that POSS can be more efficient in practice.



Subset Selection: k -Submodular Maximization

The subset selection problem as illustrated in Figure 14.1 is to select a subset maximizing some objective function subject to a size constraint. This chapter studies an extension of subset selection, i.e., the problem of maximizing monotone k -*submodular* functions subject to a size constraint, as illustrated in Figure 15.1, which appears in many real applications, e.g., influence maximization with k kinds of topics and sensor placement with k kinds of sensors [Ohsaka and Yoshida, 2015]. In such a problem, k pairwise disjoint subsets instead of a single subset need to be selected, and the objective functions are k -submodular. Note that although k -submodular is a generalization of submodular where $k = 1$ [Huber and Kolmogorov, 2012], the analysis on submodular function maximization can hardly be applied directly to the general k .

Studies on k -submodular function optimization start with $k = 2$, i.e., bi-submodular, including both minimization [Fujishige and Iwata, 2005, McCormick and Fujishige, 2010] and maximization [Singh et al., 2012, Ward and Živný, 2014]. Since 2012, much attention has been paid to the case of general k . The minimization of k -submodular functions is solvable in polynomial time [Thapper and Živný, 2012]. Maximizing k -submodular functions, however, is NP-hard. Ward and Živný [2014] first gave an $(1/3)$ -approximation guarantee by a deterministic greedy algorithm. This bound has been improved by Iwata et al. [2016] to $1/2$ with a randomized greedy algorithm. In addition, Iwata et al. [2016] gave a $(k/(2k-1))$ -approximation guarantee for monotone k -submodular function maximization.

Most of the aforementioned studies consider unconstrained situations. Meanwhile, real-world applications of k -submodular function maximization are often subject to a size constraint [Singh et al., 2012]. To the best of our knowledge, there is only one piece of work on maximizing monotone k -submodular functions subject to a size constraint. Ohsaka and Yoshida [2015] proved that the generalized greedy algorithm can obtain an $(1/2)$ -approximation guarantee subject to the total size constraint, which is asymptotically tight as exponential running time is required for achieving an ap-

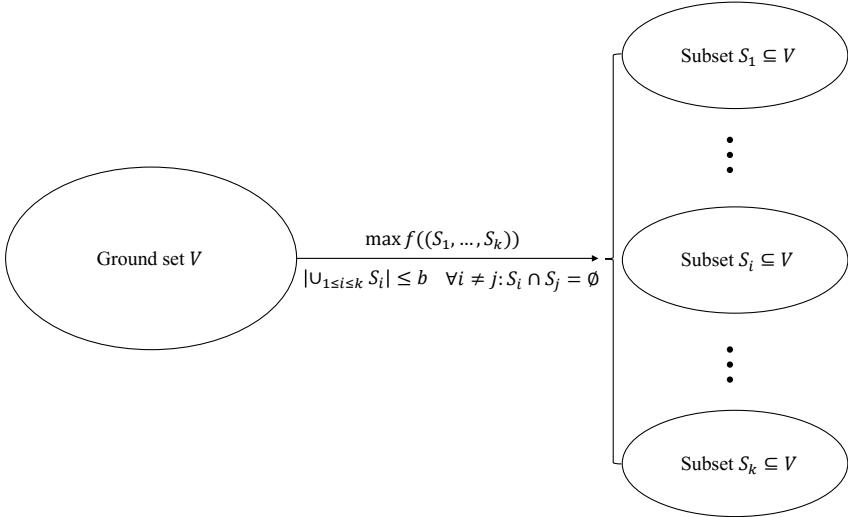


Figure 15.1. Illustration of the k -submodular maximization problem.

proximation ratio of $((k + 1)/(2k) + \epsilon)$ $\forall \epsilon > 0$ [Iwata et al., 2016]. In each iteration of the generalized greedy algorithm, a combination of one item and one subset is selected, such that putting the item into the subset can bring the largest marginal gain. They also studied the problem subject to the individual size constraint on each subset, whereas we focus on the total size constraint of the k subsets.

In this chapter, we present an algorithm POkSS [Qian et al., 2018b] based on Pareto optimization for monotone k -submodular function maximization subject to a size constraint. The idea of POkSS is similar to that of POSS. It first reformulates the original constrained optimization problem into a bi-objective optimization problem maximizing the given objective and minimizing the size simultaneously, then solves the problem by a simple MOEA combined with randomized local search, and finally selects the best feasible solution from the generated non-dominated solution set. We first theoretically analyze the performance of POkSS, and prove that POkSS can achieve the $(1/2)$ -approximation guarantee in polynomial time, reaching the asymptotically tight bound obtained by the generalized greedy algorithm [Ohsaka and Yoshida, 2015]. We also compare POkSS with the generalized greedy algorithm on three representative monotone k -submodular problems, i.e., influence maximization with k kinds of topics, information coverage maximization with k kinds of topics and sensor placement with k kinds of sensors. For each problem, we give an instance and prove that POkSS can find an optimal solution whereas the generalized greedy algorithm cannot. The empirical results on these three applications exhibit the excellent performance of POkSS.

The rest of this chapter is organized as follows. Section 15.1 introduces the concerned problem. Sections 15.2 to 15.4 present the POKSS algorithm, theoretical analysis and empirical study, respectively. Section 15.5 concludes this chapter.

15.1 Monotone k -Submodular Function Maximization

Given a finite non-empty set $V = \{v_1, v_2, \dots, v_n\}$ and a positive integer k , we study the functions $f : (k+1)^V \rightarrow \mathbb{R}$ defined on k disjoint subsets of V , where $(k+1)^V = \{(S_1, \dots, S_k) \mid S_i \subseteq V, \forall i \neq j : S_i \cap S_j = \emptyset\}$. Note that (S_1, \dots, S_k) can be represented naturally by a vector $\mathbf{s} \in \{0, 1, \dots, k\}^n$, where the j -th value $s_j = i$ means that $v_j \in S_i$, and $s_j = 0$ means that v_j does not appear in any subset; that is, $S_i = \{v_j \mid s_j = i\}$. In this chapter, $\mathbf{s} \in \{0, 1, \dots, k\}^n$ and its corresponding k disjoint subsets (S_1, \dots, S_k) will not be distinguished for notational convenience.

For $\mathbf{x} = (X_1, \dots, X_k)$ and $\mathbf{y} = (Y_1, \dots, Y_k)$, we define $\mathbf{x} \sqsubseteq \mathbf{y}$ if $\forall i : X_i \subseteq Y_i$. The monotonicity and k -submodularity are then defined as follows. When $k = 1$, k -submodular is just submodular.

Definition 15.1 (Monotone). A function $f : (k+1)^V \rightarrow \mathbb{R}$ is monotone if $\forall \mathbf{x} \sqsubseteq \mathbf{y} : f(\mathbf{x}) \leq f(\mathbf{y})$.

Definition 15.2 (k -Submodular). A function $f : (k+1)^V \rightarrow \mathbb{R}$ is k -submodular if $\forall \mathbf{x}, \mathbf{y}$:

$$f(\mathbf{x}) + f(\mathbf{y}) \geq f(\mathbf{x} \sqcap \mathbf{y}) + f(\mathbf{x} \sqcup \mathbf{y}), \quad (15.1)$$

where $\mathbf{x} \sqcap \mathbf{y} = (X_1 \cap Y_1, \dots, X_k \cap Y_k)$, and

$$\mathbf{x} \sqcup \mathbf{y} = ((X_1 \cup Y_1) \setminus \cup_{i \neq 1} (X_i \cup Y_i), \dots, (X_k \cup Y_k) \setminus \cup_{i \neq k} (X_i \cup Y_i)). \quad (15.2)$$

We then give some concepts in Definitions 15.3-15.6. The support of a solution $\mathbf{s} = (S_1, \dots, S_k)$ is the union of its k disjoint subsets, i.e., $\text{supp}(\mathbf{s}) = \cup_{i \in [k]} S_i$. For the vector representation $\mathbf{s} \in \{0, 1, \dots, k\}^n$, $S_i = \{v_j \mid s_j = i\}$, and thus $\text{supp}(\mathbf{s})$ can be represented as $\{v_j \mid s_j > 0\}$. Intuitively, the orthant submodularity means that the marginal gain by adding a single item into a solution decreases as the k subsets contained by the solution extend. The pairwise monotonicity is weaker than the monotonicity, because if a function is monotone, it must be pairwise monotone. In [Ward and Živný, 2014], the k -submodularity is proved to be equivalent to both the orthant submodularity and the pairwise monotonicity.

Definition 15.3 (Support). For a solution $\mathbf{s} = (S_1, \dots, S_k)$, the support $\text{supp}(\mathbf{s})$ is the union of its k disjoint subsets, i.e.,

$$\text{supp}(\mathbf{s}) = \cup_{i \in [k]} S_i. \quad (15.3)$$

Definition 15.4 (Marginal Gain/Loss). For a function $f : (k+1)^V \rightarrow \mathbb{R}$ and a solution $s = (S_1, \dots, S_k)$, the marginal gain $\Delta_{v,i}^+ f(s)$ with respect to $v \notin \text{supp}(s)$ and $i \in [k]$ is the increment on f by adding v into S_i , i.e.,

$$\Delta_{v,i}^+ f(s) = f(S_1, \dots, S_{i-1}, S_i \cup v, S_{i+1}, \dots, S_k) - f(S_1, \dots, S_k), \quad (15.4)$$

and the marginal loss $\Delta_{v,i}^- f(s)$ with respect to $v \in S_i$ is the decrement on f by deleting v from S_i , i.e.,

$$\Delta_{v,i}^- f(s) = f(S_1, \dots, S_k) - f(S_1, \dots, S_{i-1}, S_i \setminus v, S_{i+1}, \dots, S_k). \quad (15.5)$$

Definition 15.5 (Orthant Submodular). A function $f : (k+1)^V \rightarrow \mathbb{R}$ is orthant submodular if $\forall x \sqsubseteq y, v \notin \text{supp}(y)$ and $i \in [k]$,

$$\Delta_{v,i}^+ f(x) \geq \Delta_{v,i}^+ f(y). \quad (15.6)$$

Definition 15.6 (Pairwise Monotone). A function $f : (k+1)^V \rightarrow \mathbb{R}$ is pairwise monotone if $\forall s, v \notin \text{supp}(s)$ and $i, j \in [k]$ with $i \neq j$,

$$\Delta_{v,i}^+ f(s) + \Delta_{v,j}^+ f(s) \geq 0. \quad (15.7)$$

The concerned problem in Definition 15.7 is to select k disjoint subsets $s \in \{0, 1, \dots, k\}^n$ which maximizes a monotone k -submodular function f with an upper limit on $|\text{supp}(s)|$. Without loss of generality, assume that f is normalized, i.e., $f(\mathbf{0}) = 0$. Ohsaka and Yoshida [2015] have recently proved that the generalized greedy algorithm can obtain the asymptotically tight $(1/2)$ -approximation guarantee. As shown in Algorithm 15.1, the generalized greedy algorithm iteratively selects a combination (v, i) with the largest improvement on f . Note that $s(v)$ denotes the value of s on the item v , i.e., $v \in S_{s(v)}$. Ohsaka and Yoshida [2015] also studied the problem subject to the individual size constraint, i.e., $\forall i \in [k] : |S_i| \leq b_i$. Here, we focus on the total size constraint, i.e., $|\text{supp}(s)| = |\cup_{i \in [k]} S_i| \leq b$.

Definition 15.7 (Monotone k -Submodular Function Maximization Subject To A Size Constraint). Given a set of items $V = \{v_1, v_2, \dots, v_n\}$, a monotone k -submodular objective function $f : \{0, 1, \dots, k\}^n \rightarrow \mathbb{R}^{0+}$ and a budget $b \in [n]$, to find

$$s^* = \arg \max_{s \in \{0, 1, \dots, k\}^n} f(s) \quad \text{s.t.} \quad |\text{supp}(s)| \leq b. \quad (15.8)$$

We give three concrete applications, generalized directly from the three applications of the subset selection problem with monotone submodular objective functions, i.e., influence maximization, information coverage maximization and sensor placement.

As presented in Section 14.1.2, influence maximization is to identify a set of influential users in social networks. Let a directed graph $G = (V, E)$

Algorithm 15.1 Generalized Greedy Algorithm

Input: $V = \{v_1, v_2, \dots, v_n\}$; monotone k -submodular function $f : \{0, 1, \dots, k\}^n \rightarrow \mathbb{R}^{0+}$; budget $b \in [n]$

Output: solution $\mathbf{s} \in \{0, 1, \dots, k\}^n$ with $|\text{supp}(\mathbf{s})| = b$

Process:

- 1: let $t = 0$ and $\mathbf{s} = \mathbf{0}$;
- 2: **while** $t < b$ **do**
- 3: $(v, i) = \arg \max_{v \in V \setminus \text{supp}(\mathbf{s}), i \in [k]} \Delta_{v,i}^+ f(\mathbf{s})$;
- 4: let $\mathbf{s}(v) = i$ and $t = t + 1$
- 5: **end while**
- 6: **return** \mathbf{s}

represent a social network, where each node is a user and each edge $(u, v) \in E$ has a probability $p_{u,v}$ representing the strength of influence from user u to v . Given a budget b , the goal is to find a subset of V with at most b nodes such that the expected number of nodes activated by propagating from this selected subset is maximized [Kempe et al., 2003]. As the user-to-user influence usually depends on some topic, Barbieri et al. [2012] introduced the topic-aware model, where each edge (u, v) has a probability vector $(p_{u,v}^1, \dots, p_{u,v}^k)$ representing the influence strength from u to v on each topic. Let $\mathbf{s} \in \{0, 1, \dots, k\}^n$ represent an assignment of topics to n users, where $s_j = i$ means that user j is with topic i . Thus, S_i contains all nodes with topic i . The k -topic IC model propagates from S_i using probabilities $p_{u,v}^i$ independently for each topic. The set of nodes activated by propagating from S_i is denoted as $A(S_i)$, which is a random variable. Then, influence maximization with k kinds of topics is shown in Definition 15.8, which maximizes the expected total number of nodes that get activated in at least one propagation process. The objective function has been proved to be monotone and k -submodular [Ohsaka and Yoshida, 2015].

Definition 15.8 (Influence Maximization with k Topics). *Given a directed graph $G = (V, E)$ with $|V| = n$, edge probabilities $p_{u,v}^i$ where $(u, v) \in E$ and $i \in [k]$, and a budget b , to find*

$$\mathbf{s}^* = \arg \max_{\mathbf{s} \in \{0, 1, \dots, k\}^n} \mathbb{E} [\lvert \cup_{i \in [k]} A(S_i) \rvert] \quad \text{s.t.} \quad |\text{supp}(\mathbf{s})| \leq b. \quad (15.9)$$

As presented in Section 14.1.3, information coverage maximization is to maximize the expected number of both active and informed nodes [Wang et al., 2015]. Similarly, information coverage maximization with k kinds of topics shown in Definition 15.9 is to maximize the expected total number of nodes that get activated or informed in at least one propagation process. Let $AI(S_i)$ denote the set of active and informed nodes by propagating from S_i . $\mathbb{E}[\lvert AI(S_i) \rvert]$ is monotone and submodular [Wang et al., 2015], and thus, $\mathbb{E}[\lvert \cup_{i \in [k]} AI(S_i) \rvert]$ is monotone and k -submodular, the proof of which is the same as that for influence maximization [Ohsaka and Yoshida, 2015].

Definition 15.9 (Information Coverage Maximization with k Topics). Given a directed graph $G = (V, E)$ with $|V| = n$, edge probabilities $p_{u,v}^i$ where $(u, v) \in E$ and $i \in [k]$, and a budget b , to find

$$\mathbf{s}^* = \arg \max_{\mathbf{s} \in \{0,1,\dots,k\}^n} \mathbb{E} [\lvert \cup_{i \in [k]} AI(S_i) \rvert] \quad \text{s.t.} \quad |\text{supp}(\mathbf{s})| \leq b. \quad (15.10)$$

As presented in Section 14.1.4, the sensor placement problem is to decide where to place a limited number of sensors such that the uncertainty is mostly reduced. Assume that there are k kinds of sensors, and let $\mathbf{s} \in \{0, 1, \dots, k\}^n$ represent a placement of sensors on n locations $V = \{v_1, v_2, \dots, v_n\}$, where $s_j = i$ means that location v_j is installed with the i -th kind of sensor. Let O_j^i denote a random variable representing the observations collected from location v_j with sensor i . Let $U = \{O_j^i \mid j \in [n], i \in [k]\}$. Then, sensor placement with k kinds of topics is to maximize the entropy of $\{O_j^{s_j} \mid s_j > 0\}$ using at most b sensors, as shown in Definition 15.10. The objective function has been proved to be monotone and k -submodular [Ohsaka and Yoshida, 2015].

Definition 15.10 (Sensor Placement with k Sensors). Given n locations $V = \{v_1, v_2, \dots, v_n\}$, k kinds of sensors and a budget b , to find

$$\mathbf{s}^* = \arg \max_{\mathbf{s} \in \{0,1,\dots,k\}^n} H(\{O_j^{s_j} \mid s_j > 0\}) \quad \text{s.t.} \quad |\text{supp}(\mathbf{s})| \leq b, \quad (15.11)$$

where O_j^i denotes a random variable representing the observations collected from location v_j by installing sensor i .

15.2 The POkSS Algorithm

Based on Pareto optimization, we present an algorithm for the problem of maximizing monotone k -submodular functions subject to a size constraint, called POkSS. The POkSS algorithm first reformulates the original problem Eq. (15.8) into a bi-objective maximization problem

$$\arg \max_{\mathbf{s} \in \{0,1,\dots,k\}^n} (f_1(\mathbf{s}), f_2(\mathbf{s})), \quad (15.12)$$

where

$$f_1(\mathbf{s}) = \begin{cases} -\infty, & |\text{supp}(\mathbf{s})| \geq 2b \\ f(\mathbf{s}), & \text{otherwise} \end{cases}, \quad f_2(\mathbf{s}) = -|\text{supp}(\mathbf{s})|. \quad (15.13)$$

In other words, POkSS maximizes the original objective f and minimizes the size of $\text{supp}(\mathbf{s})$ simultaneously. To compare two solutions \mathbf{s} and \mathbf{s}' in the bi-objective setting, the domination relationship in Definition 1.2 is used.

Algorithm 15.2 POkSS Algorithm

Input: $V = \{v_1, v_2, \dots, v_n\}$; monotone k -submodular function $f : \{0, 1, \dots, k\}^n \rightarrow \mathbb{R}^{0+}$; budget $b \in [n]$

Parameter: the number T of iterations

Output: solution s with $|supp(s)| \leq b$

Process:

- 1: let $s = \mathbf{0}$ and $P = \{s\}$;
- 2: let $t = 0$;
- 3: **while** $t < T$ **do**
- 4: select a solution s from P uniformly at random;
- 5: $s' = Mutation(s)$;
- 6: **if** $\#z \in P$ such that $z \succ s'$ **then**
- 7: $P = (P \setminus \{z \in P \mid s' \succeq z\}) \cup \{s'\}$;
- 8: $Q = SLS(f, b, s')$;
- 9: **for** each $q \in Q$
- 10: **if** $\#z \in P$ such that $z \succ q$ **then**
- 11: $P = (P \setminus \{z \in P \mid q \succeq z\}) \cup \{q\}$
- 12: **end if**
- 13: **end for**
- 14: **end if**
- 15: $t = t + 1$
- 16: **end while**
- 17: **return** $\arg \max_{s \in P, |supp(s)| \leq b} f_1(s)$

After transformation, a simple MOEA with mutation only is employed to solve the reformulated bi-objective maximization problem. The employed MOEA is similar to the GSEMO algorithm, except using a special initial solution $\mathbf{0}$ and a mutation operator over $\{0, 1, \dots, k\}^n$. As described in Algorithm 15.2, it starts from the all-zeros solution and then iteratively tries to improve the solutions in the population P in lines 3-16. In each iteration, a new solution s' is generated by mutating an archived solution s selected from the current P in lines 4-5; if s' is not dominated by any previously archived solution in line 6, it will be added into P , and meanwhile those previously archived solutions weakly dominated by s' will be removed from P in line 7. To utilize local information well to improve the efficiency, a stochastic local search (SLS) procedure is incorporated to improve the newly included solution s' in line 8. SLS shown in Algorithm 15.3 adopts the random sampling technique [Mirzaoleiman et al., 2015], and performs a sequence of greedy local moves. In each iteration, it first selects a random subset R of items and then adds one item from R with the largest gain or deletes one item from R with the smallest loss. The addition or deletion depends on the relationship between $|supp(s')|$ and b . The generated solutions by SLS are then used to update P in lines 9-13. The choice of the sample size $|R|$ in each iteration of SLS will be clear in theoretical analysis.

Algorithm 15.3 SLS

Input: monotone k -submodular function $f : \{0, 1, \dots, k\}^n \rightarrow \mathbb{R}^{0+}$; budget $b \in [n]$; solution $\mathbf{s} \in \{0, 1, \dots, k\}^n$

Output: set of solutions from $\{0, 1, \dots, k\}^n$

Process:

```

1: let  $Q = \emptyset$  and  $j = |\text{supp}(\mathbf{s})|$ ;
2: if  $j < b$  then
3:   while  $|\text{supp}(\mathbf{s})| < b$  do
4:      $R \leftarrow \frac{n - |\text{supp}(\mathbf{s})|}{b - |\text{supp}(\mathbf{s})|} \log(2(b - j))$  items uniformly sampled from  $V \setminus \text{supp}(\mathbf{s})$ 
        with replacement;
5:      $(v, i) = \arg \max_{v \in R, i \in [k]} \Delta_{v,i}^+ f(\mathbf{s})$ ;
6:     let  $\mathbf{s}(v) = i$  and  $Q = Q \cup \{\mathbf{s}\}$ 
7:   end while
8: else if  $j > b$  then
9:   while  $|\text{supp}(\mathbf{s})| > b$  do
10:     $R \leftarrow \frac{b+1}{|\text{supp}(\mathbf{s})|-b}$  items uniformly sampled from  $\text{supp}(\mathbf{s})$  with replacement;
11:     $v = \arg \min_{v \in R} \Delta_{v,\mathbf{s}(v)}^- f(\mathbf{s})$ ;
12:    let  $\mathbf{s}(v) = 0$  and  $Q = Q \cup \{\mathbf{s}\}$ 
13:  end while
14: end if
15: return  $Q$ 

```

Definition 15.11 (Mutation). Given a solution $\mathbf{s} \in \{0, 1, \dots, k\}^n$, the mutation operator generates a solution \mathbf{s}' by independently flipping each position of \mathbf{s} with probability $1/n$, where the flipping on one position changes the current value to a different one selected uniformly at random. In other words,

$$\forall j \in [n] : s'_j = \begin{cases} s_j & \text{with probability } 1 - 1/n; \\ i & \text{otherwise,} \end{cases} \quad (15.14)$$

where i is chosen from $\{0, 1, \dots, k\} \setminus \{s_j\}$ uniformly at random.

POkSS repeats for T iterations. The value of T is a parameter, which is able to influence the quality of the produced solution. Their relationship will be analyzed in the theoretical analysis, and we will use the theoretically derived T value in the experiments. After running T iterations, the solution with the largest f_1 value satisfying the size constraint in P is selected as the final solution in line 17. Note that the largest f_1 value corresponds to the largest value on the original objective f .

In the bi-objective transformation, the goal of setting f_1 to $-\infty$ is to exclude overly infeasible solutions, the size of whose support is at least $2b$. These infeasible solutions with $f_1 = -\infty \wedge f_2 \leq -2b$ are dominated by any feasible solution, e.g., the empty solution with $f_1 = 0$ and $f_2 = 0$, and therefore never introduced into the population P .

15.3 Theoretical Analysis

First, we theoretically study the general performance of POkSS. We prove its general approximation bound in Theorem 15.1. Note that the $(1/2)$ -approximation ratio achieved by POkSS is asymptotically tight, because exponential running time is required for achieving an approximation ratio of $((k+1)/(2k)+\epsilon) \forall \epsilon > 0$ [Iwata et al., 2016]. The proof is inspired from that of Theorem 3.1 in [Ohsaka and Yoshida, 2015]. If the running time is allowed to be infinite, POkSS can eventually find an optimal solution, because the employed mutation operator in Definition 15.11 is a global search operator leading to a positive probability of generating any solution in each iteration.

Theorem 15.1. *For maximizing a monotone k -submodular function subject to a size constraint, POkSS with $\mathbb{E}[T] \leq 8eb$ finds a solution s with $|supp(s)| \leq b$ and $f(s) \geq OPT/2$.*

Proof. Assume that the population P always contains at least one solution s such that $|supp(s)| < b$ and

$$f(s) \geq OPT - f(z) \quad (15.15)$$

for some solution z with $s \sqsubseteq z$ and $|supp(z)| = b$. By selecting s in line 4 of Algorithm 15.2 and flipping no position in line 5, s is regenerated. Because s is previously archived in P , no solution in P can dominate s . Thus, the condition of line 6 is satisfied, and s will go into the SLS procedure in line 8.

We then claim that with probability at least $1/2$, SLS on s can generate a solution with the $(1/2)$ -approximation guarantee. Considering $|supp(s)| < b$, Algorithm 15.3 will perform lines 3-7, which iteratively adds one item into s until $|supp(s)| = b$. In the first iteration, the sampled R in line 4 is denoted as $R^{(1)}$, the selected item from R and its value in line 5 are denoted as $v^{(1)}$ and $i^{(1)}$, respectively, and the generated solution in line 6 is denoted as $s^{(1)}$. We define $z^{(1/2)}$ as a vector obtained from z by assigning 0 to one item v' in $S^{(1)} = supp(z) \setminus supp(s)$. Note that $s \sqsubseteq z$, and thus, $supp(s) \subseteq supp(z)$. Assume that $R^{(1)} \cap S^{(1)} \neq \emptyset$. If $v^{(1)} \in R^{(1)} \cap S^{(1)}$, $v' = v^{(1)}$; otherwise, v' is an arbitrary item in $R^{(1)} \cap S^{(1)}$. According to line 5, $(v^{(1)}, i^{(1)})$ is the combination increasing f the most, and thus, we have

$$\Delta_{v', z(v')}^+ f(s) \geq f(s^{(1)}) - f(s). \quad (15.16)$$

As $s \sqsubseteq z^{(1/2)}$ and $v' \notin supp(z^{(1/2)})$, we can apply the orthant submodularity, i.e., Eq. (15.6), to derive

$$\Delta_{v', z(v')}^+ f(s) \geq \Delta_{v', z(v')}^+ f(z^{(1/2)}) = f(z) - f(z^{(1/2)}). \quad (15.17)$$

Combining Eqs. (15.16) and (15.17), we have

$$f(s^{(1)}) - f(s) \geq f(z) - f(z^{(1/2)}). \quad (15.18)$$

We define $\mathbf{z}^{(1)}$ as a vector obtained from $\mathbf{z}^{(1/2)}$ by letting $\mathbf{z}^{(1/2)}(v^{(1)}) = i^{(1)}$. Note that $|supp(\mathbf{z}^{(1)})| = b$. Due to the monotonicity of f , $f(\mathbf{z}^{(1)}) \geq f(\mathbf{z}^{(1/2)})$. Thus, we have

$$f(\mathbf{s}^{(1)}) - f(\mathbf{s}) \geq f(\mathbf{z}) - f(\mathbf{z}^{(1)}). \quad (15.19)$$

Using the same analysis procedure, we have $f(\mathbf{s}^{(i)}) - f(\mathbf{s}^{(i-1)}) \geq f(\mathbf{z}^{(i-1)}) - f(\mathbf{z}^{(i)})$ for the i -th iteration. Note that lines 4-6 will repeat for $l = b - |supp(\mathbf{s})|$ iterations, and $\mathbf{z}^{(l)} = \mathbf{s}^{(l)}$. Let $\mathbf{s}^{(0)} = \mathbf{s}$ and $\mathbf{z}^{(0)} = \mathbf{z}$. We have

$$\begin{aligned} f(\mathbf{s}^{(l)}) - f(\mathbf{s}) &= \sum_{i=1}^l (f(\mathbf{s}^{(i)}) - f(\mathbf{s}^{(i-1)})) \\ &\geq \sum_{i=1}^l (f(\mathbf{z}^{(i-1)}) - f(\mathbf{z}^{(i)})) = f(\mathbf{z}) - f(\mathbf{z}^{(l)}). \end{aligned} \quad (15.20)$$

By combining $\mathbf{z}^{(l)} = \mathbf{s}^{(l)}$ with Eq. (15.20), we have

$$f(\mathbf{s}^{(l)}) \geq \frac{f(\mathbf{s}) + f(\mathbf{z})}{2} \geq \frac{OPT}{2}, \quad (15.21)$$

where the last inequality holds by Eq. (15.15).

The above analysis relies on the assumption that

$$\forall i \in [l] : R^{(i)} \cap S^{(i)} \neq \emptyset. \quad (15.22)$$

Next we show that our choice of the sample size $|R^{(i)}|$ makes Eq. (15.22) hold with probability at least $1/2$. It can be seen that $|S^{(i)}| = |supp(\mathbf{z}^{(i-1)}) \setminus supp(\mathbf{s}^{(i-1)})| = b - |supp(\mathbf{s}^{(i-1)})|$, as $|supp(\mathbf{z}^{(i-1)})| = b$ and $supp(\mathbf{s}^{(i-1)}) \subseteq supp(\mathbf{z}^{(i-1)})$. According to the procedure of generating $R^{(i)}$ in line 4 of Algorithm 15.3, we have

$$\begin{aligned} P(R^{(i)} \cap S^{(i)} = \emptyset) &= \left(1 - \frac{b - |supp(\mathbf{s}^{(i-1)})|}{n - |supp(\mathbf{s}^{(i-1)})|}\right)^{\frac{n - |supp(\mathbf{s}^{(i-1)})|}{b - |supp(\mathbf{s}^{(i-1)})|} \log(2l)} \\ &\leq \frac{1}{2l}, \end{aligned} \quad (15.23)$$

where Eq. (15.23) holds by $(1 - 1/m)^m \leq 1/e$. The assumption Eq. (15.22) then holds with probability at least $1 - l \cdot (1/(2l)) = 1/2$ by the union bound. Thus, our claim holds.

Once such a solution $\mathbf{s}^{(l)}$ is generated by SLS, it will be used to update the population P in lines 9-13 of Algorithm 15.2; this makes P always contain a solution $\mathbf{y} \succeq \mathbf{s}^{(l)}$, i.e.,

$$f(\mathbf{y}) \geq f(\mathbf{s}^{(l)}) \geq OPT/2, \quad |supp(\mathbf{y})| \leq |supp(\mathbf{s}^{(l)})| = b. \quad (15.24)$$

Thus, to find a solution with the $(1/2)$ -approximation guarantee, the required number of times to regenerate s in line 5 of Algorithm 15.2 is at most 2 in expectation. Let P_{\max} denote the largest size of the population P during the running of POkSS. The probability of selecting s in line 4 is at least $1/P_{\max}$ due to uniform selection. From Definition 15.11, the probability of flipping no position of s in line 5 is $(1 - 1/n)^n \geq 1/(2e)$. Thus, the probability of regenerating s in each iteration is at least $1/(2eP_{\max})$, and then the expected number of iterations is at most $2eP_{\max}$. Because the solutions in P are incomparable, each value of one objective can correspond to at most one solution in P . Considering $f_1 = -\infty$, any solution with $|\text{supp}(\cdot)| \geq 2b$ is excluded from P , and thus, $|\text{supp}(\cdot)|$ can only belong to $\{0, 1, \dots, 2b - 1\}$. Thus, $P_{\max} \leq 2b$, implying that the expected number of iterations is at most $2 \cdot (2e \cdot 2b) = 8eb$ until achieving the $(1/2)$ -approximation guarantee.

The initial solution $\mathbf{0}$ satisfies Eq. (15.15) by letting z be an optimal solution. Furthermore, it will always be in P , because it has the largest f_2 value 0 and no other solution can weakly dominate it. Thus, our assumption holds and then the theorem holds. \square

Thus, POkSS can generally achieve the $(1/2)$ -approximation ratio. Previous studies have shown that the generalized greedy algorithm can also achieve this asymptotically tight bound [Ohsaka and Yoshida, 2015]. We next compare POkSS with the generalized greedy algorithm on the applications of influence maximization, information coverage maximization and sensor placement. For each application, we give an instance where POkSS can perform better than the generalized greedy algorithm.

For influence maximization with k kinds of topics, we analyze Example 15.1 where each user has an equal influence probability on each topic. We prove in Theorem 15.2 that POkSS can find a global optimal solution whereas the generalized greedy algorithm cannot. The intuition of the proof is that the generalized greedy algorithm easily gets trapped in a local optimal solution, whereas POkSS can efficiently find an infeasible solution with size $b + 1$, from which backward SLS, i.e., lines 9-13 of Algorithm 15.3, can generate a global optimal solution. Let a_i be a random variable such that $a_i = 1$ if the node v_i is activated in the propagation process and $a_i = 0$ otherwise. In the proof, the objective function in Definition 15.8 is equivalently computed by $\mathbb{E}[\sum_{i=1}^n a_i] = \sum_{i=1}^n \mathbb{E}[a_i]$, where the equality holds by the linearity of expectation.

Example 15.1. The parameters of influence maximization with k kinds of topics in Definition 15.8 are set as: the graph $G = (V, E)$ is shown in Figure 15.2 where each edge has a probability vector $(1/k, \dots, 1/k)$, and the budget $b = 2$.

Theorem 15.2. *For Example 15.1, POkSS with $\mathbb{E}[T] = O(bn)$ finds a global optimal solution, whereas the generalized greedy algorithm cannot.*

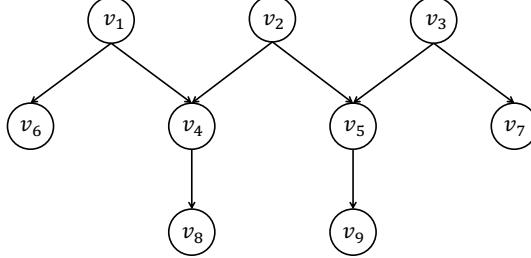


Figure 15.2. A social network graph, where each edge has a probability vector $(1/k, \dots, 1/k)$.

Proof. We first analyze $f(s)$ with $|\text{supp}(s)| \leq b = 2$. Note that s_j is the value on the node v_j . For $|\text{supp}(s)| = 1$, the solution $(0, i, 0, \dots, 0)$ with $i \in [k]$ has the largest f value $1 + 2/k + 2/k^2$, which is calculated by $\mathbb{E}[a_2] = 1$, $\mathbb{E}[a_4] = \mathbb{E}[a_5] = 1/k$, $\mathbb{E}[a_8] = \mathbb{E}[a_9] = 1/k^2$ and $\mathbb{E}[a_1] = \mathbb{E}[a_3] = \mathbb{E}[a_6] = \mathbb{E}[s_7] = 0$. For s with $s_2 > 0$ and $|\text{supp}(s)| = 2$, we can derive that, $\forall i, j \in [k]$:

$$\begin{aligned} f((j, i, 0, \dots, 0)) &= f((0, i, j, 0, \dots, 0)) \\ &= \begin{cases} 2 + \frac{4}{k} + \frac{2}{k^2} - \frac{1}{k^3} & \text{if } i = j \\ 2 + \frac{4}{k} + \frac{2}{k^2} - \frac{1}{k^4} & \text{if } i \neq j \end{cases}, \end{aligned} \quad (15.25)$$

$$\begin{aligned} f((0, i, 0, j, 0, \dots, 0)) &= f((0, i, 0, 0, j, 0, \dots, 0)) \\ &= \begin{cases} 2 + \frac{2}{k} + \frac{1}{k^2} & \text{if } i = j \\ 2 + \frac{2}{k} + \frac{2}{k^2} - \frac{1}{k^3} & \text{if } i \neq j \end{cases}, \end{aligned} \quad (15.26)$$

$$\text{otherwise, } f(s) \leq 2 + \frac{2}{k} + \frac{2}{k^2}. \quad (15.27)$$

It can be verified that $s_{\text{global}} = (i, 0, j, 0, \dots, 0)$ is a global optimal solution with objective value $2 + 4/k + 2/k^2$.

Based on the above analysis, the generalized greedy algorithm first finds $(0, i, 0, \dots, 0)$, and then goes to $s_{\text{local}} = (j, i, 0, \dots, 0)$ or $(0, i, j, 0, \dots, 0)$, where $i \neq j$. Thus, it can hardly find a global optimal solution.

For POkSS, we first show that it can efficiently find s_{local} . The required number of iterations is denoted by T_1 . Note that the initial solution 0 will always exist in P . By selecting 0 in line 4 of Algorithm 15.2 and flipping no position, occurring with probability at least $(1/(2b)) \cdot (1 - 1/n)^n \geq 1/(4eb)$, 0 is regenerated in line 5. Because 0 cannot be dominated by any solution, it will go into the SLS procedure, which generates s_{local} with probability $\Omega(1)$, because the random sample R in line 4 of SLS can cover any specific element with probability $\Omega(1)$. We pessimistically assume that s_{global} is not found, as the goal is to derive an upper bound of the number of iterations for finding s_{global} . Then, s_{local} will enter into P and always exist in P , because it has

the second largest f value among the solutions with $|supp(s)| = 2$. Thus, we have $\mathbb{E}[T_1] = O(b)$.

After that, by selecting s_{local} in line 4 and flipping the only 0 value in its first three positions to a value $l \neq i, j$ in line 5, occurring with probability at least $(1/(2b)) \cdot (1/n)(1-1/n)^{n-1}((k-2)/k)$, a solution $s^* = (j, i, l, 0, \dots, 0)$ or $(l, i, j, 0, \dots, 0)$ is generated. Such a solution has the largest f value among the solutions with $|supp(s)| = 3$, and thus, no solution can dominate it and s^* will go into the SLS procedure. As $|supp(s^*)| = b+1$, lines 10-12 of Algorithm 15.3 will be performed once. If the random sample R in line 10 covers v_2 , occurring with probability $1 - (1 - 1/(b+1))^{b+1} \geq 1 - 1/e$, s_2^* will be set to 0, leading to the smallest loss. Thus, s_{global} has been found. Denote the number of iterations in this phase as T_2 . We then have $\mathbb{E}[T_2] \leq 2ebn(k/(k-2)) \cdot (e/(e-1)) = O(bn)$.

By combining the above two phases, the expected number of iterations for POKSS finding s_{global} is at most $\mathbb{E}[T_1] + \mathbb{E}[T_2] = O(bn)$. \square

For information coverage maximization with k kinds of topics, we consider Example 15.2, which also uses the social network graph in Figure 15.2. The objective function in Definition 15.9 can be computed by $\sum_{i=1}^n \mathbb{E}[a_i]$, where $a_i = 1$ if the node v_i is activated or informed in the propagation process and $a_i = 0$ otherwise. We prove in Theorem 15.3 that POKSS performs better than the generalized greedy algorithm.

Example 15.2. The parameters of information coverage maximization with k kinds of topics in Definition 15.9 are set as: the graph $G = (V, E)$ is shown in Figure 15.2 where each edge has a probability vector $(1/k, \dots, 1/k)$, and the budget $b = 2$.

Theorem 15.3. *For Example 15.2, POKSS with $\mathbb{E}[T] = O(bn)$ finds a global optimal solution, whereas the generalized greedy algorithm cannot.*

For sensor placement with k kinds of sensors, we analyze Example 15.3 where only 4 locations with a specific kind of sensor can have different observations. Note that O_j^i denotes the observations from location v_j by installing the i -th kind of sensor. Theorem 15.4 shows that POKSS is better than the generalized greedy algorithm. The objective function, i.e., entropy, in Definition 15.10 is calculated using the observed frequency.

Example 15.3. The parameters of sensor placement with k kinds of sensors in Definition 15.10 are set as: the budget $b = 3$, and the observations collected from n locations by installing k kinds of sensors are

$$\begin{aligned} O_1^1 &= \{1, 1, 1, 2, 1, 2, 3, 3\}, \quad O_2^2 = \{1, 1, 1, 1, 2, 2, 2, 2\}, \\ O_3^3 &= \{1, 1, 2, 2, 1, 1, 2, 2\}, \quad O_4^4 = \{1, 2, 1, 2, 1, 2, 1, 2\}, \end{aligned} \tag{15.28}$$

and for any other $i \in [k], j \in [n] : O_j^i = \{1, 1, 1, 1, 1, 1, 1, 1\}$.

Theorem 15.4. For Example 15.3, POkSS with $\mathbb{E}[T] = O(kbn)$ finds a global optimal solution, whereas the generalized greedy algorithm cannot.

The proofs of Theorems 15.3 and 15.4 are provided in Appendix A.8, the intuition of which is similar to that of Theorem 15.2.

15.4 Empirical Study

In the above theoretical analyses, we have proved that POkSS can generally achieve the asymptotically tight approximation bound, which was obtained by the generalized greedy algorithm, and can perform better than the generalized greedy algorithm on some artificial instances. Here, we conduct experiments on influence maximization, information coverage maximization and sensor placement to examine the actual performance of POkSS on real-world data sets.

Because the generalized greedy algorithm is the best existing algorithm both theoretically and empirically [Ohsaka and Yoshida, 2015], we compare POkSS with it. The number T of iterations of POkSS is set to $\lfloor 8eb \rfloor$ as suggested by Theorem 15.1. We test the budget b from 5 to 10. As POkSS is a randomized algorithm, we repeat the running for 10 times independently and report the average f values. In the following, the experimental results on the three applications will be reported, respectively.

15.4.1 Influence Maximization

We first test the special case of $k = 1$, i.e., each edge on the network has one propagation probability rather than one probability vector. We use 12 real-world data sets,[†] the details of which are shown in Table 15.1. The data set *p2p-Gnutella04* is collected from the Gnutella peer-to-peer file sharing network from August 4 2002, and other *p2p-Gnutella* data sets are collected similarly. The data sets *ca-HepPh* and *ca-HepTh* are collected from the e-print arXiv, which cover collaborations between authors who submitted papers to High Energy Physics - Phenomenology category and Theory category in the period from January 1993 to April 2003, respectively. The *ego-Facebook* data set is collected from survey participants using Facebook.app. The last data set *weibo* is crawled from Weibo.com, a Chinese microblogging site. On each network, the propagation probability of one edge from node i to j is estimated by $\text{weight}(i, j)/\text{indegree}(j)$, as widely used in [Chen et al., 2009b, Goyal et al., 2011].

For the general case $k \geq 2$, i.e., each edge on the network has one propagation probability vector of length k , we use a real-world data set^{††} collected from the social news website Digg over one month in 2009. It contains two tables, i.e., the friendship links between users and the user votes

[†]<http://snap.stanford.edu/data/index.html>.

^{††}<http://www.isi.edu/~lerman/downloads/digg2009.html>.

Table 15.1. Data sets for influence maximization.

Data set	Type	#nodes	#edges
<i>p2p-Gnutella04</i>	Directed	10,879	39,994
<i>p2p-Gnutella05</i>	Directed	8,846	31,839
<i>p2p-Gnutella06</i>	Directed	8,717	31,525
<i>p2p-Gnutella08</i>	Directed	6,301	20,777
<i>p2p-Gnutella09</i>	Directed	8,114	26,013
<i>p2p-Gnutella24</i>	Directed	26,518	65,369
<i>p2p-Gnutella25</i>	Directed	22,687	54,705
<i>p2p-Gnutella30</i>	Directed	36,682	88,328
<i>ca-HepPh</i>	Undirected	12,008	118,521
<i>ca-HepTh</i>	Undirected	9,877	25,998
<i>ego-Facebook</i>	Undirected	4,039	88,234
<i>weibo</i>	Directed	10,000	162,371

on news stories [Hogg and Lerman, 2012]. After preprocessing, a directed graph with 3,523 nodes and 90,244 edges is generated. We set the number of topics $k = 2, 3, \dots, 9$, respectively, and use the method in [Barbieri et al., 2012] to estimate the edge probabilities on each topic from the user votes.

For estimating the objective function of influence maximization in Definition 15.8, i.e., the expected number of nodes that get activated, we simulate the diffusion process for 30 times independently and use the average as an estimation; this implies that the objective function evaluation is noisy. For the final output solutions of the algorithms, we average over 10,000 times for more accurate estimation. Because the behavior of the generalized greedy algorithm is randomized under noise, we repeat its running for 10 times independently and report the average f values.

The results plotted in Figures 15.3 and 15.4 show that POKSS performs consistently better than the generalized greedy algorithm on all data sets, supporting our theoretical results. Figure 15.5 depicts the improvement ratio from the generalized greedy algorithm, i.e., $(f_{POkSS} - f_{Greedy})/f_{Greedy}$, at $b = 10$, which can exceed 33% and 14% for $k = 1$ and $k \geq 2$, respectively, at best. It can also be observed from Figures 15.3 and 15.4 that the gap between POKSS and the generalized greedy algorithm has the trend of increasing with b at most cases. This may be because larger b tends to lead to more complicated problem landscapes, making the generalized greedy algorithm easier get trapped by local optima.

15.4.2 Information Coverage Maximization

To compare POKSS with the generalized greedy algorithm on the task of information coverage maximization, the 12 real-world data sets in Table 15.1 and the *Digg* data set are also used. For estimating the objective function of information coverage maximization in Definition 15.9, i.e., the expected

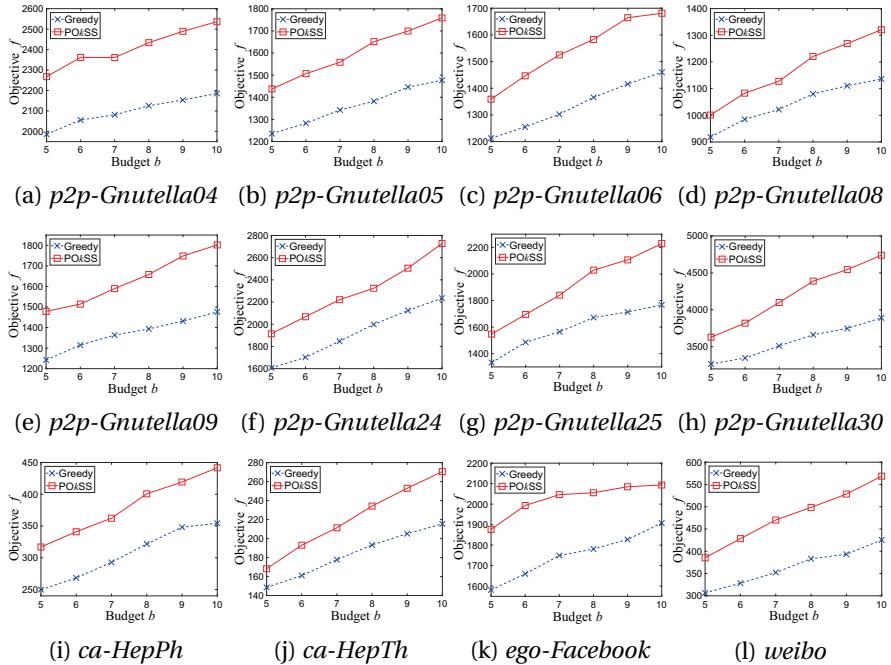


Figure 15.3. Influence maximization with $k = 1$ on the 12 data sets in Table 15.1. The objective f : the average number of active nodes (the larger, the better).

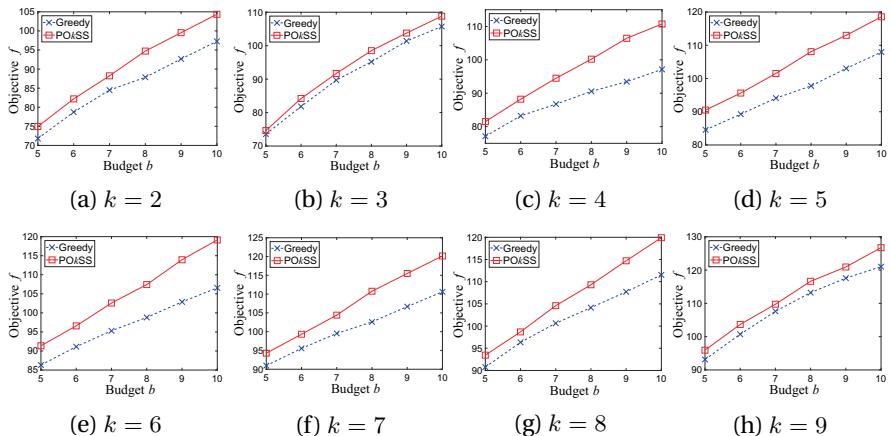


Figure 15.4. Influence maximization with $k \geq 2$ on the data set *Digg*. The objective f : the average number of active nodes (the larger, the better).

number of nodes that get activated or informed, we use the same process as that is used in the experiments of influence maximization.

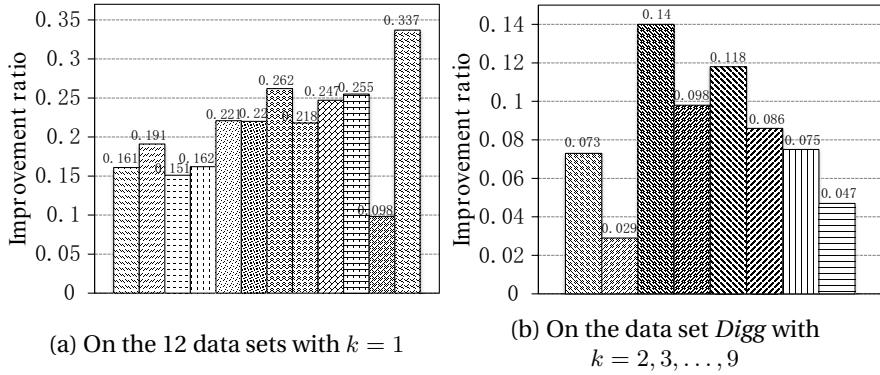


Figure 15.5. Average improvement ratio from the generalized greedy algorithm, i.e., $(f_{POkSS} - f_{Greedy})/f_{Greedy}$, at $b = 10$ for influence maximization.

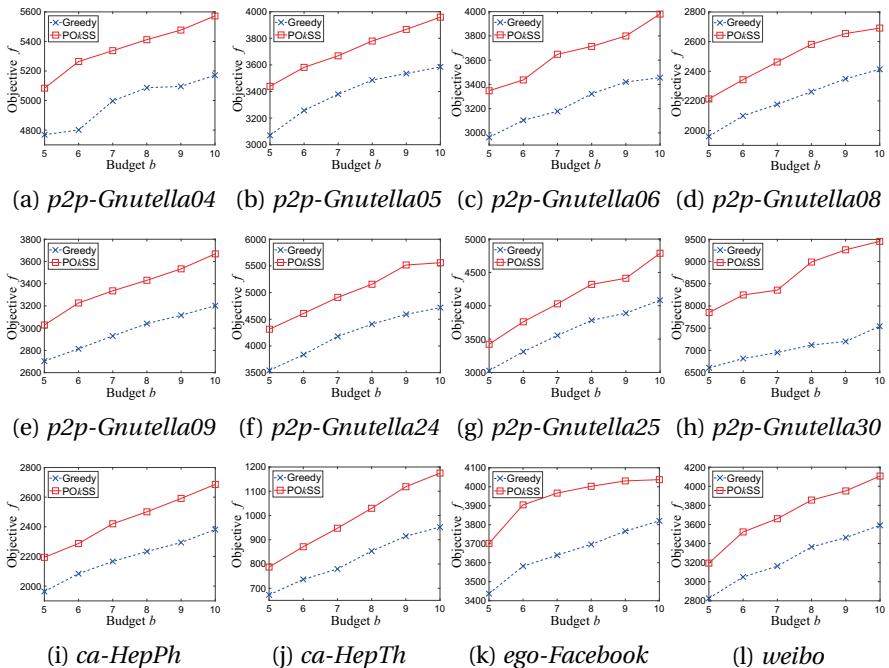


Figure 15.6. Information coverage maximization with $k = 1$ on the 12 data sets in Table 15.1. The objective f : the average number of active nodes and informed nodes (the larger, the better).

The results plotted in Figures 15.6 and 15.7 show that POkSS always performs better than the generalized greedy algorithm. Note that on the data set *ego-Facebook* in Figure 15.6(k), POkSS almost finds an optimal solution at $b = 10$, as the objective value has reached the largest possible value, i.e.,

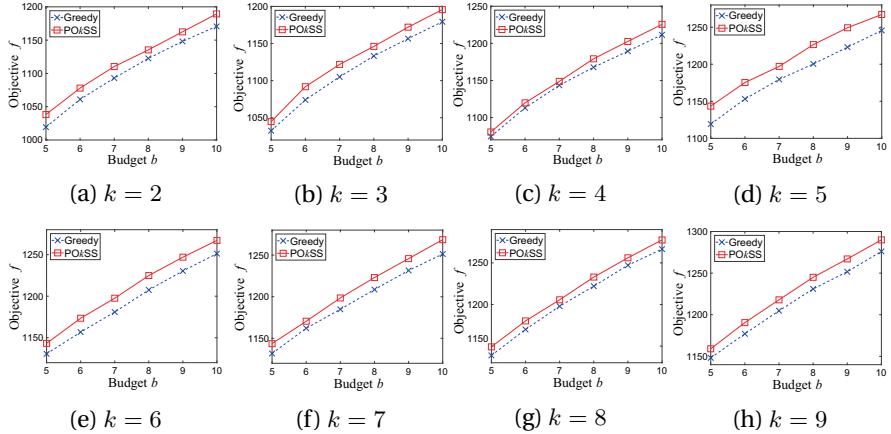


Figure 15.7. Information coverage maximization with $k \geq 2$ on the data set *Digg*. The objective f : the average number of active nodes and informed nodes (the larger, the better).

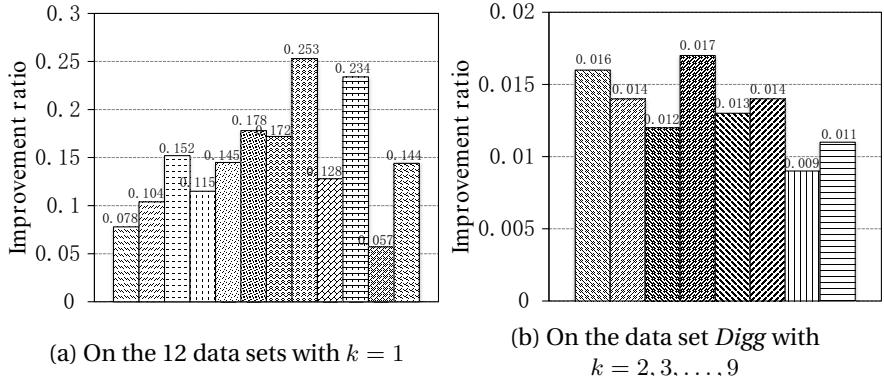


Figure 15.8. Average improvement ratio from the generalized greedy algorithm, i.e., $(f_{POkSS} - f_{Greedy})/f_{Greedy}$, at $b = 10$ for information coverage maximization.

the total number 4,039 of nodes of this network. The improvement ratio from the generalized greedy algorithm at $b = 10$ is shown in Figure 15.8. It can be seen that the improvement ratio can exceed 25% for $k = 1$ at best. For $k \geq 2$, the improvement ratio is relatively low, i.e., between 1% and 2%.

15.4.3 Sensor Placement

For the comparison on the application of sensor placement, we use a real-world data set^{†††} collected from sensors installed at 55 locations of the Intel Berkeley Research lab between February 28 and April 5, 2004. The light,

^{†††}<http://db.csail.mit.edu/labdata/labdata.html>.

temperature, voltage and humidity measures are extracted and their values are discretized into 5, 12, 6 and 13 bins with equal size, respectively. Thus, we have 4 kinds of sensors, i.e., $k = 4$, and 55 locations, i.e., $n = 55$. The problem is to select b locations and the corresponding sensors such that the entropy is maximized. We compare the performance of POKSS and the generalized greedy algorithm at $k = 1, 2, 3, 4$, respectively. For $k = 1$, we use only the light sensor at each location; for $k = 2$, the light and temperature sensors can be selected at each location; for $k = 3$, we use the light, temperature and voltage sensors; for $k = 4$, all four kinds of sensors are used. Note that the objective function, i.e., entropy, of sensor placement in Definition 15.10 is calculated using the observed frequency.

The results are plotted in Figure 15.9. It can be seen that POKSS is better than the generalized greedy algorithm for any k and b . We also calculate the improvement ratios of POKSS from the generalized greedy algorithm at $b = 10$, which are 2%, 1%, 1.2% and 0.5% for $k = 1, 2, 3, 4$, respectively.

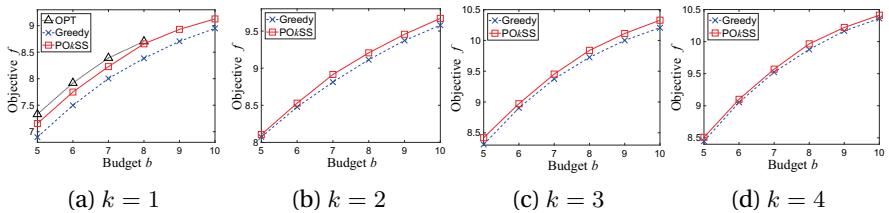


Figure 15.9. Sensor placement on the data set collected from the Intel Berkeley Research lab. The objective f : the entropy (the larger, the better).

15.4.4 Discussion

Considering the running time in the number of objective function evaluations, the generalized greedy algorithm needs $O(kbn)$ time, whereas POKSS needs $8eb$ iterations times $O(kn \log^2 b)$ time per iteration, i.e., the worst case of SLS. Note that the lazy evaluation technique [Minoux, 1978] is not considered for the fairness of comparison. Thus, POKSS is slower by a factor of $8e \log^2 b$. As this is a theoretical upper bound for POKSS to be good, we record the actual time until POKSS achieves better performance than the generalized greedy algorithm. Table 15.2 shows the results at $b = 10$ for influence maximization and information coverage maximization. It can be seen that POKSS takes only at most 5%, i.e., $5.2/(8e \log^2 10)$, of the theoretical time to achieve better performance. For sensor placement, the running time of the generalized greedy algorithm at $b = 10$ for $k = 1, 2, 3, 4$ is 505, 1,010, 1,515 and 2,020, respectively; the running time of POKSS until finding a solution better than that found by the generalized greedy algorithm is

Table 15.2. Running time comparison in the number of objective function evaluations for influence maximization and information coverage maximization at $b = 10$, where the columns of POkSS-IM and POkSS-ICM record the running time of POkSS until finding a better solution than the generalized greedy algorithm for these two tasks, respectively. Note that the running time of the generalized greedy algorithm is fixed for each data set. The number in brackets denotes the ratio between the running time of POkSS and the generalized greedy algorithm.

Data set	Greedy	POkSS-IM	POkSS-ICM	Data set	Greedy	POkSS-IM	POkSS-ICM
<i>p2p-Gnutella04</i>	108,745	126,976 (1.2)	192,145 (1.8)	<i>p2p-Gnutella25</i>	226,825	225,421 (1.0)	472,909 (2.1)
<i>p2p-Gnutella05</i>	88,415	137,236 (1.6)	171,623 (1.9)	<i>p2p-Gnutella30</i>	366,755	754,672 (2.1)	383,159 (1.0)
<i>p2p-Gnutella06</i>	87,125	205,176 (2.4)	79,976 (0.9)	<i>ca-HepPh</i>	120,035	147,153 (1.8)	180,425 (1.5)
<i>p2p-Gnutella08</i>	62,965	89,184 (1.4)	63,717 (1.0)	<i>ca-HepTh</i>	98,725	151,776 (1.5)	114,486 (1.2)
<i>p2p-Gnutella09</i>	81,095	53,645 (0.7)	87,705 (1.1)	<i>ego-Facebook</i>	40,345	47,462 (1.2)	55,697 (1.4)
<i>p2p-Gnutella24</i>	265,135	900,287 (3.4)	402,142 (1.5)	<i>weibo</i>	99,955	176,296 (1.8)	130,161 (1.3)
Digg	Greedy	POkSS-IM	POkSS-ICM	Digg	Greedy	POkSS-IM	POkSS-ICM
$k = 2$	70,370	148,621 (2.1)	144,585 (2.1)	$k = 6$	211,110	228,134 (1.1)	692,729 (3.3)
$k = 3$	105,555	544,573 (5.2)	400,764 (3.8)	$k = 7$	246,295	532,098 (2.2)	941,425 (3.8)
$k = 4$	140,740	161,975 (1.2)	499,061 (3.5)	$k = 8$	281,480	541,230 (2.0)	1,397,520 (5.0)
$k = 5$	175,925	345,494 (2.0)	325,537 (1.9)	$k = 9$	316,665	1,083,952 (3.4)	1,290,150 (4.1)

499, 3,625, 3,718 and 15,401, respectively. Thus, the ratios between the running time of POkSS and the generalized greedy algorithm are 1.0, 3.6, 2.5 and 7.6, respectively. Compared with the theoretical ratio $8e \log^2 b$, POkSS takes only at most 7%, i.e., $7.6/(8e \log^2 10)$, of the theoretical time to be better than the generalized greedy algorithm. These observations are expected, because the theoretical upper bound is the worst case running time derived based on some pessimistic assumptions.

The above results have shown that POkSS is consistently better than the generalized greedy algorithm. However, the improvement ratio can be quite different, depending on concrete k -submodular applications and data sets. For example, the improvement ratio at $b = 10$ for influence maximization can exceed 33%, whereas that for sensor placement is at most 2%. The relatively small performance improvement of POkSS in some situations may be because the generalized greedy algorithm has already performed very well. For example, for sensor placement with $k = 1$ where the improvement ratio of POkSS is empirically low, Sharma et al. [2015] proved that the generalized greedy algorithm can achieve a nearly optimal solution, also validated by experiments here. It can be seen from Figure 15.9(a) that the curve of the generalized greedy algorithm is close to that of OPT, the optimal function value calculated by exhaustive enumeration. Note that OPT is calculated only for $b = 5, 6, 7, 8$ due to the computational time limit. The approximation ratio is 94.1%, 94.7%, 95.4% and 96.3%, respectively, implying that the generalized greedy algorithm achieves a nearly optimal solution. Thus, these observations disclose that POkSS can bring a performance improvement even when the generalized greedy algorithm is nearly optimal.

Compared with the generalized greedy algorithm, POkSS has two specific characteristics: the mutation operator and the domination-based comparison. The mutation operator in Definition 15.11 is a global search operator, allowing POkSS to produce new solutions by flipping any positions. Note that the generalized greedy algorithm can flip only one position with the value 0 in each iteration. The domination relation can be used for comparing two solutions, due to the transformation of the original problem into a bi-objective one. By the domination-based comparison, POkSS naturally maintains several solutions in the population, whereas the generalized greedy algorithm maintains only one solution. These two characteristics may make POkSS have a better ability of avoiding local optima.

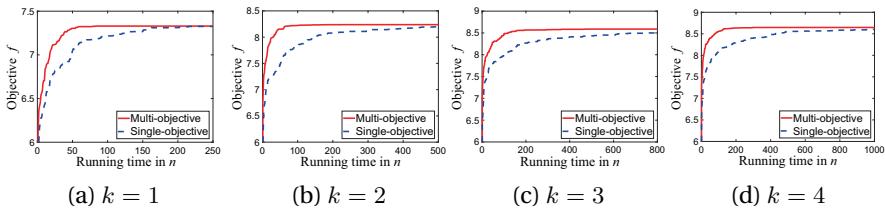


Figure 15.10. The objective f vs. the running time on sensor placement with $b = 5$, where “multi-objective” and “single-objective” denote the algorithm in the multi-objective and single-objective optimization setting, respectively. The objective f : the entropy (the larger, the better).

To validate the effectiveness of the bi-objective transformation, we conduct experiments on sensor placement with $b = 5$. We compare POkSS with its counterpart in the original single-objective setting. By single-objective optimization, the domination-based comparison is replaced by a common comparison rule “superiority of feasible points” for constrained optimization as introduced in Section 12.2: two feasible solutions are compared based on their objective values, and a feasible solution is always better than an infeasible one. For the fairness of comparison, the SLS procedure is deleted in implementing these two algorithms, as it may bring a different influence in bi-objective and single-objective optimization. We plot the curves of the objective f value over the running time. The results are shown in Figure 15.10. Note that one unit on the x -axis corresponds to n number of objective function evaluations. It can be observed that by the bi-objective transformation, the algorithm can find a much better solution using the same number of objective function evaluations. This shows the advantage of the bi-objective transformation. As the running time continues to increase, the two algorithms will gradually find the same good solution. This is expected, because both of them use the global search operator, leading to a positive probability of generating any solution; they will eventually find a global optimal solution if the running time goes to infinity.

15.5 Summary

In this chapter, we study an extension of the original subset selection problem, i.e., maximizing monotone k -submodular functions subject to a size constraint. Based on Pareto optimization, we present the POkSS algorithm for this problem. We first prove that POkSS can achieve the asymptotically tight approximation bound for general cases, which was obtained by the generalized greedy algorithm; we then show that POkSS can be better than the generalized greedy algorithm on special instances. Our empirical results on the applications of influence maximization, information coverage maximization and sensor placement verify the excellent performance of POkSS. Particularly, for sensor placement with $k = 1$ where the generalized greedy algorithm achieves a nearly optimal solution, the comparison results show that POkSS can bring a performance improvement even when the generalized greedy algorithm has been almost optimal.



Subset Selection: Ratio Minimization

This chapter studies the problem of selecting a subset that minimizes the ratio of two set functions, i.e., f/g , as illustrated in Figure 16.1, which has been found useful in various applications. For example, many machine learning tasks, such as optimizing F-measure [Rijsbergen and Joost, 1974], linear discriminant analysis [McLachlan, 1992], normalized cut [Shi and Malik, 2000], etc., involve ratio optimization. Recently, Bai et al. [2016] studied the ratio minimization problem where the functions f and g are monotone and submodular, denoted as RS minimization.

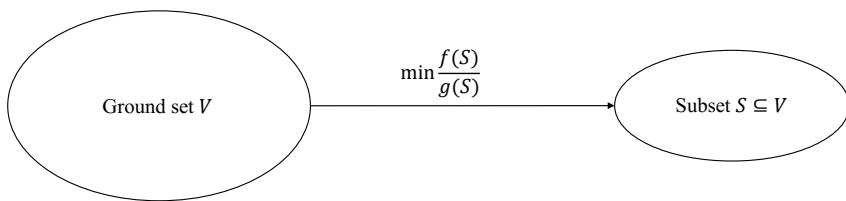


Figure 16.1. Illustration of the ratio minimization problem.

For RS minimization, several algorithms with bounded approximation guarantees have been proposed [Bai et al., 2016]. The GreedRatio algorithm iteratively selects one item to make the ratio of the marginal gain on f and g minimized. Other algorithms connect RS minimization to the problem of minimizing the difference between submodular functions [Iyer and Bilmes, 2012] or connect it to the problem of submodular optimization with submodular constraints [Iyer and Bilmes, 2013], and then apply existing techniques of the related problems. GreedRatio has been shown to obtain the best empirical performance in the application of F-measure maximization.

This chapter considers a more general situation, i.e., the function g on the denominator can be non-submodular. We first prove that GreedRatio

obtains a $\left(\frac{|S^*|}{1+(|S^*|-1)(1-\hat{\kappa}_f(S^*))} \cdot \frac{1}{\gamma_{\emptyset,|S^*|}(g)} \right)$ -approximation ratio, where S^* is an optimal subset, $\hat{\kappa}_f(S^*)$ is the curvature of f , and $\gamma_{\emptyset,|S^*|}(g)$ is the submodularity ratio of g . When g is submodular, the derived bound becomes $\frac{|S^*|}{1+(|S^*|-1)(1-\hat{\kappa}_f(S^*))}$, improving the previously known $1/(1-e^{\kappa_f-1})$ [Bai et al., 2016]. Particularly, the approximation bound is improved from $+\infty$ to $|S^*|$ for $\kappa_f = 1$, and improved from $e/(e-1)$ to 1 for $\kappa_f = 0$.

The greedy nature of GreedRatio results in an efficient fixed time algorithm, but meanwhile may limit its performance. We then present an anytime algorithm PORM [Qian et al., 2017b] based on Pareto optimization, which can take more time to find better solutions. PORM first reformulates the original problem f/g into a bi-objective optimization problem which minimizes f and maximizes g simultaneously, then solves the problem by a simple MOEA, and finally selects the solution with the smallest ratio from the maintained set of solutions. Compared with *single-bit forward search* by GreedRatio, PORM can perform *backward search*, *multi-path search* and *multi-bit search*, which may help alleviate the issue of getting trapped in local optima. Our theoretical results for PORM disclose that, within reasonable time, PORM can achieve the same general approximation guarantee as GreedRatio; in a case of F-measure maximization, PORM can escape local optima by backward search, multi-path search or multi-bit search to find a global optimal solution, whereas GreedRatio cannot. Experimental results on F-measure maximization exhibit the excellent performance of PORM.

The rest of this chapter is organized as follows. Section 16.1 introduces the concerned problem. Sections 16.2 to 16.4 present the PORM algorithm, theoretical analysis and empirical study, respectively. Section 16.5 concludes this chapter.

16.1 Ratio Minimization of Monotone Submodular Functions

Given a finite set $V = \{v_1, v_2, \dots, v_n\}$, we study the functions $f : 2^V \rightarrow \mathbb{R}$ defined on subsets of V . The curvature in Definition 16.1 characterizes how close a monotone submodular set function f is to modularity. It can be verified that $\hat{\kappa}_f(S) \leq \kappa_f(S) \leq \kappa_f$. Note that f is modular iff $\kappa_f = 0$.

Definition 16.1 (Curvature [Conforti and Cornuéjols, 1984, Vondrák, 2010, Iyer et al., 2013]). Let f be a monotone submodular set function. The total curvature of f is

$$\kappa_f = 1 - \min_{v \in V: f(v) > 0} \frac{f(V) - f(V \setminus v)}{f(v)}. \quad (16.1)$$

The curvature with respect to a set $S \subseteq V$ is

$$\kappa_f(S) = 1 - \min_{v \in S: f(v) > 0} \frac{f(S) - f(S \setminus v)}{f(v)}, \quad (16.2)$$

and an alternative notion is

$$\hat{\kappa}_f(S) = 1 - \frac{\sum_{v \in S} (f(S) - f(S \setminus v))}{\sum_{v \in S} f(v)}. \quad (16.3)$$

We study the problem in Definition 16.2 minimizing the ratio of a monotone submodular function f and a monotone function g . We can assume that $\forall v \in V : f(v) > 0$, because $\forall v$ with $f(v) = 0$, we can simply add it into the final subset without increasing the ratio. Note that only minimization is considered because maximizing f/g is equivalent to minimizing g/f .

Definition 16.2 (Ratio Minimization of Monotone Submodular Functions). *Given a monotone submodular function $f : 2^V \rightarrow \mathbb{R}^{0+}$ and a monotone function $g : 2^V \rightarrow \mathbb{R}^{0+}$, to find*

$$S^* = \arg \min_{\emptyset \subset S \subseteq V} \frac{f(S)}{g(S)}. \quad (16.4)$$

Maximizing the F-measure in information retrieval is a special instance of this problem with g being submodular. Given a bipartite graph $G(V, W, E)$, where V is a set of objects, W is a set of words and each edge $(v, w) \in E$ means that the object v contains the word w , define the function $\Gamma : 2^V \rightarrow 2^W$ as $\forall S \subseteq V : \Gamma(S) = \{w \in W \mid \exists v \in S : (v, w) \in E\}$, i.e., $\Gamma(S)$ is the set of words contained by the objects in S . Then, the information retrieval problem of finding a subset of objects which exactly cover a target set $O \subseteq W$ of words can be formulated as maximizing the F-measure of the coverage on O , as shown in Definition 16.3. Both $|\Gamma(S) \cap O|$ and $(|O| + |\Gamma(S)|)$ are monotone and submodular.

Definition 16.3 (F-measure Maximization). *Given a bipartite graph $G(V, W, E)$ and a target subset $O \subseteq W$, to find*

$$S^* = \arg \max_{\emptyset \subset S \subseteq V} \left(F(S) = \frac{2|\Gamma(S) \cap O|}{|O| + |\Gamma(S)|} \right). \quad (16.5)$$

Bai et al. [2016] have studied the problem of minimizing the ratio of monotone submodular functions, i.e., the function g in Definition 16.2 is submodular. They proved that the GreedRatio algorithm can obtain an $(1/(1 - e^{\kappa_f} - 1))$ -approximation ratio, and conduct experiments to show that GreedRatio achieves the best performance on F-measure maximization. As shown in Algorithm 16.1, GreedRatio iteratively selects one item v such that the ratio of the marginal gain on f and g by adding v is minimized.

Algorithm 16.1 GreedRatio Algorithm**Input:** monotone submodular functions $f, g : 2^V \rightarrow \mathbb{R}^{0+}$ **Output:** subset $S \subseteq V$ **Process:**

- 1: let $S_0 = \emptyset, R = V$ and $i = 0$;
- 2: **while** $R \neq \emptyset$ **do**
- 3: $v \in \arg \min_{v \in R} \frac{f(S_i \cup v) - f(S_i)}{g(S_i \cup v) - g(S_i)}$;
- 4: $S_{i+1} = S_i \cup v$;
- 5: $R = \{v \mid g(S_{i+1} \cup v) - g(S_{i+1}) > 0\}$;
- 6: $i = i + 1$
- 7: **end while**
- 8: **return** S_{i^*} with $i^* = \arg \min_i f(S_i)/g(S_i)$

We theoretically analyze the performance of GreedRatio for our concerned general problem, i.e., g is not necessarily submodular. We prove its general approximation bound in Theorem 16.1. Let S^* be an optimal solution with the minimum size, i.e., $f(S^*)/g(S^*) = \text{OPT}$ and $|S^*| = \min\{|S| \mid f(S)/g(S) = \text{OPT}\}$. The intuition of the proof is that the best single item v^* obtains the desired approximation bound as shown in Lemma 16.2, and the subset output by GreedRatio in line 8 of Algorithm 16.1 satisfies $f(S_{i^*})/g(S_{i^*}) \leq f(v^*)/g(v^*)$.

Lemma 16.1. [Iyer et al., 2013] *Given a monotone submodular function $f : 2^V \rightarrow \mathbb{R}^{0+}$, it holds that, $\forall S \subseteq V$:*

$$\sum_{v \in S} f(v) \leq \frac{|S|}{1 + (|S| - 1)(1 - \hat{\kappa}_f(S))} f(S). \quad (16.6)$$

Lemma 16.2. *For minimizing the ratio f/g where f is monotone submodular and g is monotone, $\exists v^* \in V$ such that*

$$\frac{f(v^*)}{g(v^*)} \leq \frac{|S^*|}{1 + (|S^*| - 1)(1 - \hat{\kappa}_f(S^*))} \frac{1}{\gamma_{\emptyset, |S^*|}(g)} \cdot \text{OPT}. \quad (16.7)$$

Proof. Let $v^* = \arg \min_{v \in V} f(v)/g(v)$. By the definition of submodularity ratio in Definition 14.1, we have

$$\begin{aligned} g(S^*) &\leq \frac{\sum_{v \in S^*} g(v)}{\gamma_{\emptyset, |S^*|}(g)} \leq \frac{1}{\gamma_{\emptyset, |S^*|}(g)} \frac{g(v^*)}{f(v^*)} \sum_{v \in S^*} f(v) \\ &\leq \frac{1}{\gamma_{\emptyset, |S^*|}(g)} \frac{g(v^*)}{f(v^*)} \cdot \frac{|S^*| \cdot f(S^*)}{1 + (|S^*| - 1)(1 - \hat{\kappa}_f(S^*))}, \end{aligned} \quad (16.8)$$

where Eq. (16.8) holds by Lemma 16.1. Thus, the lemma holds. \square

Theorem 16.1. For minimizing the ratio f/g where f is monotone submodular and g is monotone, GreedRatio finds a subset $S \subseteq V$ with

$$\frac{f(S)}{g(S)} \leq \frac{|S^*|}{1 + (|S^*| - 1)(1 - \hat{\kappa}_f(S^*))} \frac{1}{\gamma_{\emptyset, |S^*|}(g)} \cdot \text{OPT}. \quad (16.9)$$

For the special case of g being submodular, $\forall S, k : \gamma_{S,k}(g) = 1$, and thus, the obtained bound in Theorem 16.1 becomes $\frac{|S^*|}{1 + (|S^*| - 1)(1 - \hat{\kappa}_f(S^*))}$ shown in Corollary 16.1. This improves the previous known bound $1/(1 - e^{\kappa_f - 1})$ [Bai et al., 2016], because

$$\frac{|S^*|}{1 + (|S^*| - 1)(1 - \hat{\kappa}_f(S^*))} \leq \frac{1}{1 - \hat{\kappa}_f(S^*)} \leq \frac{1}{1 - \kappa_f} \leq \frac{1}{1 - e^{\kappa_f - 1}}, \quad (16.10)$$

where the first inequality holds by $\hat{\kappa}_f(S^*) \in [0, 1]$, the second inequality holds by $\hat{\kappa}_f(S^*) \leq \kappa_f(S^*) \leq \kappa_f$, and the last holds by $\kappa_f = 1 - (1 - \kappa_f) \leq e^{\kappa_f - 1}$. Particularly, the previous known bound becomes vacuous when $\kappa_f = 1$, whereas our derived bound has an upper limit $|S^*|$. When f is modular, i.e., $\kappa_f = 0$, our derived bound discloses that GreedRatio finds an optimal solution, whereas the known bound in [Bai et al., 2016] shows that GreedRatio achieves only an $(e/(e - 1))$ -approximation guarantee.

Corollary 16.1. For minimizing the ratio f/g where both f and g are monotone and submodular, GreedRatio finds a subset $S \subseteq V$ with

$$\frac{f(S)}{g(S)} \leq \frac{|S^*|}{1 + (|S^*| - 1)(1 - \hat{\kappa}_f(S^*))} \cdot \text{OPT}. \quad (16.11)$$

16.2 The PORM Algorithm

The greedy nature of GreedRatio may limit its performance. To alleviate the risk of getting trapped in local optima, we present the Pareto Optimization algorithm for Ratio Minimization (PORM). PORM reformulates the original problem Eq. (16.4) into a bi-objective maximization problem

$$\arg \max_{s \in \{0,1\}^n} (-f(s), g(s)). \quad (16.12)$$

In other words, PORM minimizes f and maximizes g simultaneously. The domination relationship in Definition 1.2 is used to compare solutions.

The procedure of PORM is described in Algorithm 16.2. It employs the GSEMO algorithm to solve the reformulated bi-objective maximization problem. Starting from a random solution in line 1, PORM iteratively tries to improve the solutions in the population P in lines 3-15. In each iteration, a new solution s' is generated by applying bit-wise mutation on an archived solution s selected from the current P in lines 4-5; if s' is not dominated by

Algorithm 16.2 PORM Algorithm

Input: monotone submodular function $f : \{0, 1\}^n \rightarrow \mathbb{R}^{0+}$; monotone function $g : \{0, 1\}^n \rightarrow \mathbb{R}^{0+}$

Parameter: the number T of iterations

Output: solution $s \in \{0, 1\}^n$

Process:

```

1: let  $s$  = a solution uniformly and randomly selected from  $\{0, 1\}^n$ ;
2: let  $P = \{s\}$  and  $t = 0$ ;
3: while  $t < T$  do
4:   select a solution  $s$  from  $P$  uniformly at random;
5:   apply bit-wise mutation on  $s$  to generate  $s'$ ;
6:   if  $\nexists z \in P$  such that  $z \succ s'$  then
7:      $P = (P \setminus \{z \in P \mid s' \succeq z\}) \cup \{s'\}$ ;
8:      $Q = \{z \in P \mid |z|_1 = |s'|_1\}$ ;
9:      $z_1 = \arg \min_{z \in Q} f(z)$ ;
10:     $z_2 = \arg \max_{z \in Q} g(z)$ ;
11:     $z_3 = \arg \min_{z \in Q} f(z)/g(z)$ ;
12:     $P = (P \setminus Q) \cup \{z_1, z_2, z_3\}$ 
13:   end if
14:    $t = t + 1$ 
15: end while
16: return  $\arg \min_{s \in P} f(s)/g(s)$ 

```

any archived solution in line 6, it will be added into P and meanwhile those archived solutions weakly dominated by s' will be removed from P in line 7.

Although the domination-based comparison makes the population P contain only incomparable solutions, the size of P can be large, which may reduce the efficiency of PORM. To control the size of P and meanwhile avoid the loss of useful information, three informative solutions are kept for each subset size $|s|_1 \in \{0, 1, \dots, n\}$ in lines 8-12: z_1 and z_2 are two boundary solutions with the smallest f value and the largest g value, respectively, and z_3 is the solution with the smallest ratio. Note that z_3 can be the same as z_1 or z_2 . Thus, $|P|$ is upper bounded by $1 + 3(n - 1) + 1 = 3n - 1$.

PORM repeats for T iterations. The value of T affects the quality of the produced solution, which will be analyzed in the theoretical analysis. After the iterations, the solution with the smallest ratio in P is selected in line 16.

Compared with GreedRatio, PORM can escape local optima by three different ways: (1) backward search which flips one bit value from 1 to 0; (2) multi-path search which maintains several incomparable solutions in the population P ; (3) multi-bit search which flips more than one 0-bits to 1-bits simultaneously. These advantages of PORM over GreedRatio will be theoretically shown.

16.3 Theoretical Analysis

We first prove the general approximation bound of PORM in Theorem 16.2, showing that PORM reaches the same approximation guarantee as GreedRatio. Note that $\mathbb{E}[T]$ depends on the f value of the initial solution, denoted as f_{init} , generated in line 1 of Algorithm 16.2 and the minimum f value $f_{\min} = \min\{f(v) \mid v \in V\}$.

The intuition of the proof is to follow the behavior of GreedRatio. In other words, the optimization process is divided into two phases: (1) starts from an initial random solution and finishes until finding the special solution 0^n , i.e., \emptyset ; (2) starts after phase (1) and finishes until finding a solution with the desired approximation guarantee. Lemma 16.3 shows the expected number of iterations of phase (1), derived by using multiplicative drift analysis in Theorem 2.4.

Lemma 16.3. *For minimizing the ratio f/g where f is monotone submodular and g is monotone, PORM with $\mathbb{E}[T] \leq en(3n-1)(1+\log(f_{\text{init}}/f_{\min}))/(\kappa_f)$ finds the solution 0^n .*

Proof. We use Theorem 2.4 for the proof. The optimization process is modeled by a Markov chain $\{\xi_t\}_{t=0}^{+\infty}$. Let $V(\xi_t) = \min\{f(s) \mid s \in P\}$ be the minimum f value of solutions in the population P after t iterations of PORM. $V(\xi_t) = 0$ implies that the solution 0^n is found, because $f(s) > 0$ for any non-empty subset s . Thus, the variable τ in Theorem 2.4 is just the number of iterations required by PORM for finding 0^n .

We examine $\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t]$. Let \hat{s} be the corresponding solution with $f(\hat{s}) = V(\xi_t)$. We first show that $V(\xi_t)$ cannot increase, i.e., $V(\xi_{t+1}) \leq V(\xi_t)$. If \hat{s} is not deleted, $V(\xi_t)$ will not increase. Note that there are two possible cases for removing \hat{s} from P . If \hat{s} is deleted in line 7 of Algorithm 16.2, the newly included solution s' must weakly dominate \hat{s} , implying $f(s') \leq f(\hat{s})$. If \hat{s} is deleted in line 12, $|\hat{s}|_1 = |s'|_1$ and $f(s')$ must be smaller than $f(\hat{s})$, because the solution in Q with the smallest f value is kept. Thus, $V(\xi_t)$ will not increase.

Next we show that $V(\xi_t)$ can decrease by flipping only one 1-bit of \hat{s} . Let P_{\max} denote the largest size of P during the optimization process. In the $(t+1)$ -th iteration, consider that \hat{s} is selected in line 4 of Algorithm 16.2, occurring with probability at least $1/P_{\max}$ due to uniform selection; in line 5, only the i -th bit of \hat{s} , i.e., \hat{s}_i , is flipped, occurring with probability $(1/n)(1-1/n)^{n-1} \geq 1/(en)$. If $\hat{s}_i = 1$, the newly generated solution $s' = \hat{s} \setminus v_i$. By the monotonicity of f , we have $f(s') = f(\hat{s} \setminus v_i) \leq f(\hat{s})$. If the inequality strictly holds, s' now has the smallest f value and will enter into P , leading to $V(\xi_{t+1}) = f(\hat{s} \setminus v_i) < V(\xi_t)$. If $f(\hat{s} \setminus v_i) = f(\hat{s})$, we have $V(\xi_{t+1}) = V(\xi_t)$; but we can still write $V(\xi_{t+1}) = f(\hat{s} \setminus v_i)$.

Thus, we have shown that $V(\xi_t)$ does not increase, and can decrease by flipping only one 1-bit of \hat{s} . We then have

$$\mathbb{E}[V(\xi_{t+1}) \mid \xi_t] \leq \sum_{i:\hat{s}_i=1} \frac{f(\hat{s} \setminus v_i)}{enP_{\max}} + \left(1 - \frac{|\hat{s}|_1}{enP_{\max}}\right) V(\xi_t), \quad (16.13)$$

leading to

$$\begin{aligned} \mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t] &= V(\xi_t) - \mathbb{E}[V(\xi_{t+1}) \mid \xi_t] \\ &\geq \sum_{i:\hat{s}_i=1} \frac{V(\xi_t) - f(\hat{s} \setminus v_i)}{enP_{\max}} = \frac{\sum_{v \in \hat{s}} (f(\hat{s}) - f(\hat{s} \setminus v))}{enP_{\max}}. \end{aligned} \quad (16.14)$$

By the definition of curvature in Definition 16.1, we have

$$1 - \hat{\kappa}_f(\hat{s}) = \frac{\sum_{v \in \hat{s}} (f(\hat{s}) - f(\hat{s} \setminus v))}{\sum_{v \in \hat{s}} f(v)} \leq \frac{\sum_{v \in \hat{s}} (f(\hat{s}) - f(\hat{s} \setminus v))}{f(\hat{s})}, \quad (16.15)$$

where the inequality holds by Eq. (14.2), i.e., $f(\hat{s}) = f(\hat{s}) - f(\emptyset) \leq \sum_{v \in \hat{s}} (f(v) - f(\emptyset)) = \sum_{v \in \hat{s}} f(v)$. Applying Eq. (16.15) to Eq. (16.14), we have

$$\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t] \geq \frac{1 - \hat{\kappa}_f(\hat{s})}{enP_{\max}} f(\hat{s}) \geq \frac{1 - \kappa_f}{en(3n - 1)} V(\xi_t), \quad (16.16)$$

where the last inequality holds by $P_{\max} \leq 3n - 1$, $V(\xi_t) = f(\hat{s})$ and $\hat{\kappa}_f(\hat{s}) \leq \kappa_f(\hat{s}) \leq \kappa_f$. This implies that the condition of Theorem 2.4 holds with $c = (1 - \kappa_f)/(en(3n - 1))$. By Theorem 2.4 and considering $V(\xi_0) = f_{\text{init}}$ and $V_{\min} = f_{\min}$, we have

$$\mathbb{E}[\tau \mid \xi_0] \leq \frac{en(3n - 1)}{1 - \kappa_f} \left(1 + \log \frac{f_{\text{init}}}{f_{\min}}\right). \quad (16.17)$$

□

Theorem 16.2. *For minimizing the ratio f/g where f is monotone submodular and g is monotone, PORM with $\mathbb{E}[T] \leq en(3n - 1)(1 + (1 + \log(f_{\text{init}}/f_{\min}))/(\kappa_f))$ finds a solution s with*

$$\frac{f(s)}{g(s)} \leq \frac{|S^*|}{1 + (|S^*| - 1)(1 - \hat{\kappa}_f(S^*))} \frac{1}{\gamma_{\emptyset, |S^*|}(g)} \cdot \text{OPT}. \quad (16.18)$$

Proof. We analyze phase (2) after finding the special solution 0^n . Note that once 0^n is found, it will always exist in the population P , because it has the smallest f value and no other solutions can dominate it. By selecting 0^n in line 4 of Algorithm 16.2 and flipping only the bit corresponding to the best single item $v^* = \arg \min_{v \in V} f(v)/g(v)$ in line 5, occurring with probability at least $(1/P_{\max}) \cdot (1/n)(1 - 1/n)^{n-1} \geq 1/(en(3n - 1))$, the new solution $s' = \{v^*\}$ is generated. Then, s' is used to update P in lines 6-13, making P

always contain a solution which either weakly dominates s' or is incomparable with s' but has a smaller ratio. In other words, P will always contain a solution s with

$$\frac{f(s)}{g(s)} \leq \frac{f(v^*)}{g(v^*)} \leq \frac{|S^*|}{1 + (|S^*| - 1)(1 - \hat{\kappa}_f(S^*))} \frac{\text{OPT}}{\gamma_{\emptyset, |S^*|}(g)}, \quad (16.19)$$

where the last inequality holds by Lemma 16.2. Thus, phase (2) requires at most $en(3n - 1)$ expected number of iterations.

By combining the two phases, the total expected number of iterations for finding a solution with a $\left(\frac{|S^*|}{1 + (|S^*| - 1)(1 - \hat{\kappa}_f(S^*))} \cdot \frac{1}{\gamma_{\emptyset, |S^*|}(g)}\right)$ -approximation guarantee is

$$\mathbb{E}[T] \leq en(3n - 1) \left(1 + \frac{1}{1 - \kappa_f} \left(1 + \log \frac{f_{\text{init}}}{f_{\min}} \right) \right). \quad (16.20)$$

□

Next we show that PORM can be better than GreedRatio in cases. By using an illustrative example of F-measure maximization, we prove in Theorem 16.3 that GreedRatio will get trapped in a local optimal solution, whereas PORM can avoid local optima by three different ways, and finally find a global optimal solution. For Example 16.1, it has a unique global optimal solution $1^{n-1}0$, i.e., $\{v_1, \dots, v_{n-1}\}$. The intuition of the proof is that GreedRatio will first select the object v_n due to the greedy nature and will be misled, whereas PORM can avoid v_n by backward search, multi-path search or multi-bit search, which will be shown in the proof, respectively.

Example 16.1. Let $V = \{v_1, v_2, \dots, v_n\}$. The function Γ and the target set O satisfy that

$$(1) \quad \forall i, j \in [n - 1] : \Gamma(v_i) \cap \Gamma(v_j) = \emptyset, |\Gamma(v_i)| = |\Gamma(v_j)| = n^2, \quad (16.21)$$

$$(2) \quad \forall i \in [n - 1] : |O \cap \Gamma(v_i)| = n^2 - 1, |O| = (n - 1)(n^2 - 1), \quad (16.22)$$

$$(3) \quad \forall i \in [n - 1] : \Gamma(v_n) \cap \Gamma(v_i) \subseteq O \cap \Gamma(v_i),$$

$$|\Gamma(v_n) \cap \Gamma(v_i)| = n + 2, |\Gamma(v_n)| = |\Gamma(v_n) \cap O| + 1. \quad (16.23)$$

Theorem 16.3. *For the F-measure maximization example, i.e., Example 16.1, PORM with $\mathbb{E}[T] \leq en(3n - 1)(2 + 2 \log n)$ finds the optimal solution $1^{n-1}0$, whereas GreedRatio cannot.*

Proof. By the definition of $F(s) = (2|\Gamma(s) \cap O|)/(|O| + |\Gamma(s)|)$, we have, $\forall s$ with $|s|_1 = i \wedge s_n = 0$, $F(s) = 2(n^2 - 1)i/(n^3 - n^2 - n + 1 + n^2i)$, which increases with i and reaches the maximum $(1 - 1/(2n^2 - 1))$ when $i = n - 1$; $\forall s$ with $|s|_1 = i \wedge s_n = 1$, $F(s) = (4n + 2 + 2(n^2 - n - 3)i)/(n^3 - n^2 + n + 2 + (n^2 - n - 2)i)$, which increases with i and reaches the maximum $(1 - 1/(2n^2 - 2n - 1 + 2/n))$ when $i = n$. Thus, $1^{n-1}0$ is the unique optimal solution.

For GreedRatio, it first selects one object v with the best ratio of the marginal gain, i.e., the largest $(2|\Gamma(v) \cap O|)/(|O| + |\Gamma(v)| - |O|)$ value. $\forall i \in [n - 1]$, we have

$$\frac{2|\Gamma(v_i) \cap O|}{|\Gamma(v_i)|} = \frac{2(n^2 - 1)}{n^2} < \frac{2(n^2 + n - 2)}{n^2 + n - 1} = \frac{2|\Gamma(v_n) \cap O|}{|\Gamma(v_n)|}, \quad (16.24)$$

implying that GreedRatio will first select v_n . From line 8 of Algorithm 16.1, the final subset output by GreedRatio will always contain v_n . Thus, GreedRatio cannot find the optimal solution $\{v_1, \dots, v_{n-1}\}$.

For the PORM algorithm, the problem of F-measure maximization is implemented as minimizing $f(s) = |O| + |\Gamma(s)|$ and maximizing $g(s) = |\Gamma(s) \cap O|$ simultaneously. We prove that PORM can find the optimal solution $\{v_1, \dots, v_{n-1}\}$ by three different ways.

[Backward search] The intuition is that PORM first efficiently finds V , and then the optimal solution can be generated by deleting v_n from V .

Let G_t equal $\max\{g(s) \mid s \in P\}$ after t iterations of Algorithm 16.2. We first use Theorem 2.4 to derive the number of iterations, denoted as T_1 , until G_t reaches the maximum $|O|$. Let $V(\xi_t) = |O| - G_t$. Then, the random variable τ in Theorem 2.4 is just T_1 , because $V(\xi_t) = 0$ is equivalent to $G_t = |O|$. Considering $\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t] = \mathbb{E}[G_{t+1} - G_t \mid G_t]$, we only need to analyze the change of G_t . Let \hat{s} be the corresponding solution with $g(\hat{s}) = G_t$. As in the proof of Lemma 16.3, we can similarly show that G_t does not decrease, and can increase by flipping only one 0-bit of \hat{s} . We then have

$$\begin{aligned} \mathbb{E}[G_{t+1} - G_t \mid G_t] &\geq \sum_{i:\hat{s}_i=0} \frac{g(\hat{s} \cup v_i) - g(\hat{s})}{enP_{\max}} = \frac{\sum_{v \in V \setminus \hat{s}} (g(\hat{s} \cup v) - g(\hat{s}))}{enP_{\max}} \\ &\geq \frac{g(V) - g(\hat{s})}{enP_{\max}} \geq \frac{V(\xi_t)}{en(3n - 1)}, \end{aligned} \quad (16.25)$$

where Eq. (16.25) holds by the submodularity of g , i.e., Eq. (14.2). This implies that the condition of Theorem 2.4 holds with $c = 1/(en(3n - 1))$. By Theorem 2.4 and considering that $V(\xi_0) = |O| - G_0 \leq |O| = (n - 1)(n^2 - 1)$ and $V_{\min} = n^2 - 1 - (n + 2)$, we have

$$\mathbb{E}[T_1] = \mathbb{E}[\tau \mid \xi_0] \leq en(3n - 1)(1 + 2 \log n). \quad (16.26)$$

From Example 16.1, a solution s with $g(s) = |\Gamma(s) \cap O| = |O|$ must contain v_1, \dots, v_{n-1} . There are two possible solutions: $1^{n-1}0$ and 1^n . To derive an upper bound of the number of iterations for finding the optimal solution $1^{n-1}0$, we pessimistically assume that $1^{n-1}0$ is not found. For the case that P contains 1^n , the optimal solution $1^{n-1}0$ can be generated by selecting 1^n in line 4 and flipping the last 1-bit in line 5, occurring with probability at least $(1/P_{\max}) \cdot (1/n)(1 - 1/n)^{n-1} \geq 1/(en(3n - 1))$. Denote the number of iterations in this phase as T_2 . We then have $\mathbb{E}[T_2] \leq en(3n - 1)$.

Therefore, the expected number of iterations for PORM to find the optimal solution $1^{n-1}0$ is

$$\mathbb{E}[T] \leq \mathbb{E}[T_1] + \mathbb{E}[T_2] \leq en(3n - 1)(2 + 2 \log n). \quad (16.27)$$

[Multi-path search] $\forall i \in [n - 1]$, let s^i denote any solution with $|s^i|_1 = i \wedge s_n^i = 0$, i.e., there are i 1s in the first $(n - 1)$ bits and the last bit is 0. The intuition is that PORM first efficiently finds the empty set 0^n , and then follows the path $s^1 \rightarrow s^2 \rightarrow \dots \rightarrow s^{n-1}$ to generate the optimal solution. Note that although s^1 is worse than the solution $0^{n-1}1$, i.e., $\{v_n\}$, on the original objective f/g , they are incomparable in the bi-objective setting, and thus, s^1 will be kept in P , allowing PORM to follow a path different from that by GreedRatio to find the optimal solution.

Let T_1 denote the number of iterations for finding the solution 0^n . Due to the fact that the $f(s)$ value increases with the number of 1-bits of s , we can derive a better upper bound for $\mathbb{E}[T_1]$ than Lemma 16.3. Let j denote the minimum number of 1-bits of solutions in P . First, j cannot increase, because the smallest f value of solutions in P cannot increase. Second, j can decrease by 1 in each iteration with probability at least $(1/P_{\max}) \cdot (j/n)(1 - 1/n)^{n-1} \geq j/(en(3n - 1))$, as it is sufficient to select a solution with j 1-bits in line 4 and flip only one of its j 1-bits in line 5. Thus, for reaching $j = 0$, we have

$$\mathbb{E}[T_1] \leq \sum_{j=1}^n \frac{en(3n - 1)}{j} \leq en(3n - 1)(1 + \log n). \quad (16.28)$$

Starting from 0^n , let T_2 denote the number of iterations for following the path $s^1 \rightarrow s^2 \rightarrow \dots \rightarrow s^{n-1}$, i.e., the optimal solution. Note that $\forall i \in [n - 1]$, s^i cannot be weakly dominated by any other solution, and there are only two different objective vectors for $|s|_1 = i$. According to the updating procedure of PORM in lines 6-13 of Algorithm 16.2, s^i will be always kept in P once it is found. The probability of $s^i \rightarrow s^{i+1}$ is at least $(1/P_{\max}) \cdot ((n - 1 - i)/n)(1 - 1/n)^{n-1} \geq (n - 1 - i)/(en(3n - 1))$, because it is sufficient to select s^i in line 4 and then flip only one of its first $(n - 1 - i)$ 0-bits. Thus,

$$\mathbb{E}[T_2] \leq \sum_{i=0}^{n-2} \frac{en(3n - 1)}{n - 1 - i} \leq en(3n - 1)(1 + \log n). \quad (16.29)$$

Combining the above two phases leads to the total expected number of iterations for finding the optimal solution:

$$\mathbb{E}[T] \leq \mathbb{E}[T_1] + \mathbb{E}[T_2] \leq en(3n - 1)(2 + 2 \log n). \quad (16.30)$$

[Multi-bit search] The intuition is that flipping two 0-bits in the first $(n - 1)$ positions of 0^n simultaneously can find the solution s^2 , which has a better f/g value than the solution s with $|s|_1 = 2 \wedge s_n = 1$ found by

GreedRatio; then the optimal solution can be found by following the path $s^2 \rightarrow s^3 \rightarrow \dots \rightarrow s^{n-1}$.

Using the same analysis as multi-path search, PORM can first find the solution 0^n in at most $en(3n - 1)(1 + \log n)$ expected number of iterations. After that, selecting 0^n in line 4 and flipping only any two 0s in its first $(n - 1)$ bits can generate the solution s^2 with probability at least $(1/P_{\max}) \cdot ((n-1)/n^2)(1 - 1/n)^{n-2} \geq (n-1)(n-2)/(2e(3n-1)n^2)$. Compared with the solution s with $|s|_1 = 2 \wedge s_n = 1$, we have

$$\frac{f(s^2)}{g(s^2)} = \frac{(n+1)n^2 - n + 1}{2n^2 - 2} < \frac{(n+1)n^2 - n - 2}{2n^2 - 5} = \frac{f(s)}{g(s)}. \quad (16.31)$$

PORM can then follow the path $s^2 \rightarrow s^3 \rightarrow \dots \rightarrow s^{n-1}$ to find the optimal solution. The total expected number of iterations is at most

$$\begin{aligned} en(3n - 1)(1 + \log n) + \frac{2e(3n - 1)n^2}{(n - 1)(n - 2)} + \sum_{i=2}^{n-2} \frac{en(3n - 1)}{n - 1 - i} \\ \leq en(3n - 1)(2 + 2 \log n). \end{aligned} \quad (16.32)$$

Taking the minimum of the expected number of iterations for finding the optimal solution by backward search, multi-path search and multi-bit search, the theorem holds. \square

16.4 Empirical Study

In this section, we conduct experiments on F-measure maximization to examine the actual performance of PORM. We compare PORM with GreedRatio, as it is the algorithm achieving the best empirical performance up to now [Bai et al., 2016]. The generalized form of F-measure is used for evaluation, i.e., $F_p(S) = (|\Gamma(S) \cap O|)/(p|O| + (1 - p)|\Gamma(S)|)$. We will test $p \in \{0.2, \dots, 0.8\}$. Note that the F-measure in Definition 16.3 is just $F_{0.5}$. For PORM, the number T of iterations is set to $\lfloor 3en^2(2 + \log |\Gamma(S_{\text{init}})|) \rfloor$, where S_{init} is the initial subset generated by PORM, as suggested by Theorem 16.2. Note that we have used the lower bound 1 for $1/(1 - \kappa_f)$.

We use synthetic (*syn-100*, *syn-1000*) as well as real-world (*ldc-100*, *ldc-1000*) data sets. For *syn-100*, a bipartite graph $G(V, W, E)$ with $|V| = |W| = 100$ is randomly generated by adding one edge between $v \in V$ and $w \in W$ independently with probability 0.05; the target set O with $|O| = 20$ is randomly chosen from W . For *syn-1000*, $|V| = |W| = 1,000$, $|O| = 100$ and each edge is added with probability 0.01. *ldc-100* and *ldc-1000* contain 100 and 1,000 English sentences, respectively, which are randomly sampled from the LDC2002E18 text data.[†] Their target sets contain 1,000 randomly chosen words. For each data set, we generate ten random instances and report

[†]<https://www.ldc.upenn.edu/>.

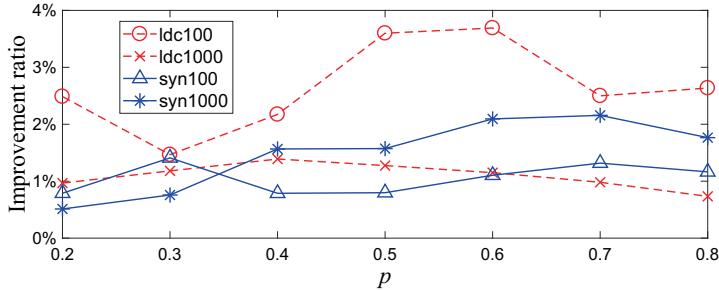


Figure 16.2. Ratio of improvement of PORM to GreedRatio.

the average results. Considering that PORM is a randomized algorithm, its running is further repeated for 10 times independently for each data set instance. Figure 16.2 shows the percentages of the solution quality that PORM improves from GreedRatio, where it can be observed that PORM is always better than GreedRatio and can have more than 3% improvement.

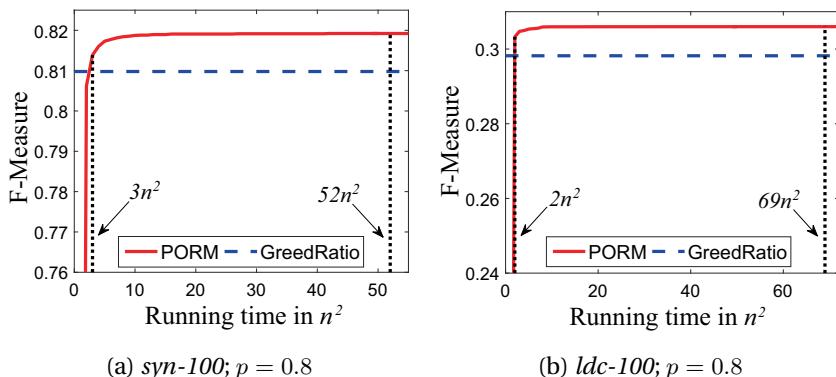


Figure 16.3. Performance vs. running time of PORM and GreedRatio.

Comparing the running time in the number of function calculations, GreedRatio takes the time in the order of n^2 ; PORM is set to use $\lfloor 3en^2(2 + \log |\Gamma(S_{\text{init}})|) \rfloor$ time according to the theoretical upper bound, i.e., the worst-case time, for PORM to be good. We empirically examine how effective PORM is in practice. By selecting GreedRatio for the baseline, we plot the curves of the F-measure over the running time for PORM on *syn-100* and *ldc-100* with $p = 0.8$, as shown in Figure 16.3. The x -axis is in n^2 , i.e., the running time of GreedRatio. It can be observed that PORM takes about only 6% ($3/52$) and 3% ($2/69$) of the worst-case running time to achieve better performance, respectively, implying that PORM can be efficient in practice.

16.5 Summary

In this chapter, we study the problem of minimizing the ratio f/g , where f is monotone submodular and g is just monotone. Such problems often appear in machine learning tasks, such as in the form of optimizing the F-measure. We prove the approximation bound of the greedy-style algorithm GreedRatio for the problem, which even improves the previously known result when g is submodular. We then present a new algorithm PORM based on Pareto optimization, and prove that PORM can achieve the same general approximation guarantee as GreedRatio, but can have a better ability of avoiding local optima. The empirical results on the application of F-measure maximization verify the excellent performance of PORM.



Subset Selection: Noise

From Chapter 10, we have gained some understanding on the influence of noise. This chapter studies the subset selection problem under noise. For subset selection, there are many practical situations where the evaluation can be noisy. For example, in the influence maximization problem, computing the influence spread objective is #P-hard [Chen et al., 2010b], and thus, is often estimated by simulating the random diffusion process [Kempe et al., 2003], which brings noise; in the sparse regression problem, only a set of limited data can be used for evaluation, which makes the evaluation noisy; more examples include maximizing information gain in graphical models [Chen et al., 2015], crowdsourced image collection summarization [Singla et al., 2016], etc.

Only a few studies on noisy subset selection have been reported, assuming monotone submodular objective functions. Under the general multiplicative noise model, i.e., the noisy objective value $f^n(S)$ is in the range of $(1 \pm \epsilon) \cdot f(S)$, it has been proved that no polynomial-time algorithm can achieve a constant approximation ratio $\forall \epsilon > 1/\sqrt{n}$, whereas the greedy algorithm can achieve an $(1 - 1/e - 16\delta)$ -approximation ratio for $\epsilon = \delta/k$ as long as $\delta < 1$ [Horel and Singer, 2016]. By assuming that $f^n(S)$ is a random variable, i.e., random noise, and the expectation of $f^n(S)$ is the true value $f(S)$, it has been shown that the greedy algorithm can achieve nearly an $(1 - 1/e)$ -approximation guarantee via uniform sampling [Kempe et al., 2003] or adaptive sampling [Singla et al., 2016]. Recently, Hassidim and Singer [2017] considered the consistent random noise model, where for each subset S , only the first evaluation is a random draw from the distribution of $f^n(S)$ and the other evaluations return the same value. For some classes of noise distribution, they provided polynomial-time algorithms with constant approximation guarantees.

In this chapter, we consider a more general situation, i.e., noisy subset selection with a monotone objective f , not necessarily submodular, for both multiplicative noise and additive noise, i.e., $f^n(S)$ is in the range of $f(S) \pm \epsilon$. First, we extend the approximation ratio of the greedy algorithm

from the submodular case [Horel and Singer, 2016] to the general situation, and slightly improve it. Then, we prove that the approximation ratio of POSS presented in Chapter 14 is nearly the same as that of the greedy algorithm. Moreover, on two maximum coverage examples, we show that POSS can have a better ability of avoiding the misleading search direction due to noise.

To improve POSS under noise, we introduce a noise-aware comparison strategy, and present the new PONSS algorithm [Qian et al., 2017a] for noisy subset selection. When comparing two solutions with similar noisy objective values, POSS selects the solution with the better observed value, whereas PONSS keeps both of them such that the risk of deleting a good solution is reduced. With some assumption such as independently and identically distributed (i.i.d.) noise distribution, we prove that PONSS can obtain an $(\frac{1-\epsilon}{1+\epsilon}(1-e^{-\gamma}))$ -approximation guarantee under multiplicative noise. Particularly for constant ϵ and the submodular case where $\gamma = 1$, PONSS achieves a constant approximation guarantee. Note that for the greedy algorithm and POSS under general multiplicative noise, only a $\Theta(1/k)$ -approximation ratio is guaranteed. We also prove the approximation guarantee of PONSS under additive noise.

We conduct experiments on influence maximization and sparse regression, two typical subset selection applications with the objective function being submodular and non-submodular, respectively. The results on real-world data sets show that POSS is better than the greedy algorithm in most cases, and PONSS clearly outperforms POSS and the greedy algorithm.

The rest of this chapter is organized as follows. Section 17.1 introduces the noisy subset selection problem and presents theoretical analysis for the greedy and POSS algorithms. Sections 17.2 to 17.4 present the PONSS algorithm, theoretical analysis and empirical study, respectively. Section 17.5 concludes this chapter.

17.1 Noisy Subset Selection

Given a finite set $V = \{v_1, v_2, \dots, v_n\}$, we study the functions $f : 2^V \rightarrow \mathbb{R}$ defined on subsets of V . The subset selection problem in Definition 14.2 is to select a subset S of V such that a given objective f is maximized with the constraint $|S| \leq b$. Monotone objective functions are considered.

In many applications of subset selection, we cannot obtain the exact objective value $f(S)$, but rather only a noisy one $f^n(S)$. In our analysis, we will study the multiplicative noise model, i.e.,

$$(1 - \epsilon) \cdot f(S) \leq f^n(S) \leq (1 + \epsilon) \cdot f(S), \quad (17.1)$$

as well as the additive noise model, i.e.,

$$f(S) - \epsilon \leq f^n(S) \leq f(S) + \epsilon. \quad (17.2)$$

17.1.1 The Greedy Algorithm

The greedy algorithm shown in Algorithm 14.1 iteratively adds one item with the largest marginal gain until b items are selected. It can achieve the best approximation ratio for many subset selection problems without noise [Nemhauser and Wolsey, 1978, Das and Kempe, 2011]. However, its performance for noisy subset selection has not been theoretically analyzed until recently. Note that under noise, the item with the largest increment on f^n rather than f is selected in each iteration of the greedy algorithm. Horel and Singer [2016] proved that for subset selection with submodular objective functions under the multiplicative noise model, the greedy algorithm finds a subset S with

$$f(S) \geq \frac{\frac{1-\epsilon}{1+\epsilon}}{1 + \frac{4b\epsilon}{(1-\epsilon)^2}} \left(1 - \left(\frac{1-\epsilon}{1+\epsilon} \right)^{2b} \left(1 - \frac{1}{b} \right)^b \right) \cdot \text{OPT}. \quad (17.3)$$

Note that their original bound in Theorem 5 of [Horel and Singer, 2016] is with respect to $f^n(S)$, and has been switched to $f(S)$ by multiplying a factor of $(1-\epsilon)/(1+\epsilon)$ according to Eq. (17.1).

By extending their analysis with the submodularity ratio, we prove in Theorem 17.1 the approximation bound of the greedy algorithm for the objective f being not necessarily submodular. Note that their analysis is based on a recursive inequality on f^n , whereas we directly use that on f , bringing a slight improvement. For the submodular case, $\gamma_{S,b} = 1$ and the bound in Theorem 17.1 changes to

$$f(S) \geq \frac{\frac{1-\epsilon}{1+\epsilon} \frac{1}{b}}{1 - \frac{1-\epsilon}{1+\epsilon} \left(1 - \frac{1}{b} \right)} \left(1 - \left(\frac{1-\epsilon}{1+\epsilon} \right)^b \left(1 - \frac{1}{b} \right)^b \right) \cdot \text{OPT}. \quad (17.4)$$

Compared with Eq. (17.3), our bound is tighter, because

$$\begin{aligned} \frac{1 - \left(\frac{1-\epsilon}{1+\epsilon} \right)^b \left(1 - \frac{1}{b} \right)^b}{1 - \frac{1-\epsilon}{1+\epsilon} \left(1 - \frac{1}{b} \right)} &= \sum_{i=0}^{b-1} \left(\frac{1-\epsilon}{1+\epsilon} \left(1 - \frac{1}{b} \right) \right)^i \\ &\geq \sum_{i=0}^{b-1} \left(\left(\frac{1-\epsilon}{1+\epsilon} \right)^2 \left(1 - \frac{1}{b} \right) \right)^i \geq \frac{1 - \left(\frac{1-\epsilon}{1+\epsilon} \right)^{2b} \left(1 - \frac{1}{b} \right)^b}{1 + \frac{4b\epsilon}{(1-\epsilon)^2}} \cdot b. \end{aligned} \quad (17.5)$$

Theorem 17.1. *For subset selection under multiplicative noise, the greedy algorithm finds a subset S with*

$$f(S) \geq \frac{\frac{1-\epsilon}{1+\epsilon} \frac{\gamma_{S,b}}{b}}{1 - \frac{1-\epsilon}{1+\epsilon} \left(1 - \frac{\gamma_{S,b}}{b} \right)} \left(1 - \left(\frac{1-\epsilon}{1+\epsilon} \right)^b \left(1 - \frac{\gamma_{S,b}}{b} \right)^b \right) \cdot \text{OPT}. \quad (17.6)$$

Proof. Let S^* be an optimal subset, i.e., $f(S^*) = \text{OPT}$. Let S_i denote the subset after the i -th iteration of the greedy algorithm. Then, we have

$$f(S^*) - f(S_i) \leq f(S^* \cup S_i) - f(S_i) \quad (17.7)$$

$$\leq \frac{1}{\gamma_{S_i,b}} \sum_{v \in S^* \setminus S_i} (f(S_i \cup v) - f(S_i)) \quad (17.8)$$

$$\leq \frac{1}{\gamma_{S_i,b}} \sum_{v \in S^* \setminus S_i} \left(\frac{1}{1-\epsilon} f^n(S_i \cup v) - f(S_i) \right) \quad (17.9)$$

$$\leq \frac{1}{\gamma_{S_i,b}} \sum_{v \in S^* \setminus S_i} \left(\frac{1}{1-\epsilon} f^n(S_{i+1}) - f(S_i) \right) \quad (17.10)$$

$$\leq \frac{b}{\gamma_{S_b,b}} \left(\frac{1+\epsilon}{1-\epsilon} f(S_{i+1}) - f(S_i) \right), \quad (17.11)$$

where Eq. (17.7) holds by the monotonicity of f , Eq. (17.8) holds by the definition of submodularity ratio and $|S^*| \leq b$, Eq. (17.9) holds by the definition of multiplicative noise, i.e., $f^n(S) \geq (1-\epsilon) \cdot f(S)$, Eq. (17.10) holds by line 3 of Algorithm 14.1, and Eq. (17.11) holds by $\gamma_{S_i,b} \geq \gamma_{S_{i+1},b}$ and $f^n(S) \leq (1+\epsilon) \cdot f(S)$. By a simple transformation, we can equivalently get

$$f(S_{i+1}) \geq \left(\frac{1-\epsilon}{1+\epsilon} \right) \left(\left(1 - \frac{\gamma_{S_b,b}}{b} \right) f(S_i) + \frac{\gamma_{S_b,b}}{b} \text{OPT} \right). \quad (17.12)$$

Based on Eq. (17.12), an inductive proof gives the approximation ratio of the returned subset S_b :

$$f(S_b) \geq \frac{\frac{1-\epsilon}{1+\epsilon} \frac{\gamma_{S_b,b}}{b}}{1 - \frac{1-\epsilon}{1+\epsilon} \left(1 - \frac{\gamma_{S_b,b}}{b} \right)} \left(1 - \left(\frac{1-\epsilon}{1+\epsilon} \right)^b \left(1 - \frac{\gamma_{S_b,b}}{b} \right)^b \right) \cdot \text{OPT}. \quad (17.13)$$

□

Theorem 17.2 shows the approximation ratio under additive noise. The proof is similar to that of Theorem 17.1, except that Eq. (17.2) is used rather than Eq. (17.1) for comparing $f(S)$ with $f^n(S)$.

Theorem 17.2. *For subset selection under additive noise, the greedy algorithm finds a subset S with*

$$f(S) \geq \left(1 - \left(1 - \frac{\gamma_{S,b}}{b} \right)^b \right) \cdot \left(\text{OPT} - \frac{2b\epsilon}{\gamma_{S,b}} \right). \quad (17.14)$$

17.1.2 The POSS Algorithm

Let a Boolean vector $s \in \{0,1\}^n$ represent a subset S of V , where $s_i = 1$ if $v_i \in S$ and $s_i = 0$ otherwise. The POSS algorithm reformulates the original problem Eq. (14.5) as a bi-objective maximization problem:

$$\arg \max_{s \in \{0,1\}^n} (f_1(s), f_2(s)), \quad (17.15)$$

$$\text{where } f_1(s) = \begin{cases} -\infty, & |s|_1 \geq 2b \\ f^n(s), & \text{otherwise} \end{cases}, \quad f_2(s) = -|s|_1. \quad (17.16)$$

In other words, POSS maximizes the original objective and minimizes the subset size simultaneously. As described in Algorithm 14.2, POSS uses a simple MOEA to optimize the bi-objective problem. After running T iterations, the solution with the largest f^n value satisfying the size constraint is selected from the maintained population P .

In Chapter 14, POSS with $\mathbb{E}[T] \leq 2eb^2n$ has been shown to achieve the same approximation ratio as the greedy algorithm for subset selection without noise. Its approximation performance under noise, however, is not yet known. Note that in Algorithm 14.2, POSS employs an isolation function I , which can be ignored if I is a constant function. We assume that I is a constant function when not explicitly mentioning it. Let $\gamma_{\min} = \min_{s:|s|_1=b-1} \gamma_{s,b}$. We prove in Theorem 17.3 the approximation ratio of POSS under multiplicative noise. Lemma 17.1 gives the relation between the f^n values of adjacent subsets, whose proof is provided in Appendix A.9.

Lemma 17.1. $\forall s \in \{0, 1\}^n$, there exists one item $v \notin s$ such that

$$f^n(s \cup v) \geq \left(\frac{1-\epsilon}{1+\epsilon} \right) \left(1 - \frac{\gamma_{s,b}}{b} \right) f^n(s) + \frac{(1-\epsilon)\gamma_{s,b}}{b} \cdot \text{OPT}. \quad (17.17)$$

Theorem 17.3. For subset selection under multiplicative noise, POSS with $\mathbb{E}[T] \leq 2eb^2n$ finds a solution s with $|s|_1 \leq b$ and

$$f(s) \geq \frac{\frac{1-\epsilon}{1+\epsilon} \frac{\gamma_{\min}}{b}}{1 - \frac{1-\epsilon}{1+\epsilon} \left(1 - \frac{\gamma_{\min}}{b} \right)} \left(1 - \left(\frac{1-\epsilon}{1+\epsilon} \right)^b \left(1 - \frac{\gamma_{\min}}{b} \right)^b \right) \cdot \text{OPT}. \quad (17.18)$$

Proof. Let J_{\max} denote the maximum value of $j \in [b]$ such that in P , $\exists s$ with $|s|_1 \leq j$ and

$$f^n(s) \geq \frac{(1-\epsilon)\frac{\gamma_{\min}}{b}}{1 - \frac{1-\epsilon}{1+\epsilon} \left(1 - \frac{\gamma_{\min}}{b} \right)} \left(1 - \left(\frac{1-\epsilon}{1+\epsilon} \right)^j \left(1 - \frac{\gamma_{\min}}{b} \right)^j \right) \cdot \text{OPT}. \quad (17.19)$$

We analyze the expected number T of iterations until $J_{\max} = b$, implying that $\exists s \in P$ satisfying that $|s|_1 \leq b$ and $f^n(s) \geq \frac{(1-\epsilon)\frac{\gamma_{\min}}{b}}{1 - \frac{1-\epsilon}{1+\epsilon} \left(1 - \frac{\gamma_{\min}}{b} \right)} \left(1 - \left(\frac{1-\epsilon}{1+\epsilon} \right)^b \left(1 - \frac{\gamma_{\min}}{b} \right)^b \right) \cdot \text{OPT}$. Because $f(s) \geq f^n(s)/(1+\epsilon)$, the desired approximation bound has been reached when $J_{\max} = b$.

J_{\max} is initially 0, as POSS starts from 0^n . Assume that currently $J_{\max} = i < b$. Let s be a corresponding solution with the value i , i.e., $|s|_1 \leq i$ and

$$f^n(s) \geq \frac{(1-\epsilon)\frac{\gamma_{\min}}{b}}{1 - \frac{1-\epsilon}{1+\epsilon} \left(1 - \frac{\gamma_{\min}}{b} \right)} \left(1 - \left(\frac{1-\epsilon}{1+\epsilon} \right)^i \left(1 - \frac{\gamma_{\min}}{b} \right)^i \right) \cdot \text{OPT}. \quad (17.20)$$

J_{\max} cannot decrease because deleting s from P in lines 7 and 8 of Algorithm 14.2 implies that s is weakly dominated by the newly generated solution s' , which must have a smaller size and a larger f^n value. By Lemma 17.1, flipping one specific 0-bit of s , i.e., adding a specific item, can generate a new solution s' , satisfying that

$$\begin{aligned} f^n(s') &\geq \left(\frac{1-\epsilon}{1+\epsilon} \right) \left(1 - \frac{\gamma_{s,b}}{b} \right) f^n(s) + \frac{(1-\epsilon)\gamma_{s,b}}{b} \cdot \text{OPT} \\ &= \frac{1-\epsilon}{1+\epsilon} f^n(s) + \left(\text{OPT} - \frac{f^n(s)}{1+\epsilon} \right) \frac{(1-\epsilon)\gamma_{s,b}}{b}. \end{aligned} \quad (17.21)$$

Note that $\text{OPT} - f^n(s)/(1+\epsilon) \geq f(s) - f^n(s)/(1+\epsilon) \geq 0$. Moreover, $\gamma_{s,b} \geq \gamma_{\min}$, because $|s|_1 < b$ and $\gamma_{s,b}$ decreases with s . Thus, we have

$$f^n(s') \geq \left(\frac{1-\epsilon}{1+\epsilon} \right) \left(1 - \frac{\gamma_{\min}}{b} \right) f^n(s) + \frac{(1-\epsilon)\gamma_{\min}}{b} \cdot \text{OPT}. \quad (17.22)$$

By applying Eq. (17.20) to Eq. (17.22), we have

$$f^n(s') \geq \frac{(1-\epsilon)\frac{\gamma_{\min}}{b}}{1 - \frac{1-\epsilon}{1+\epsilon} \left(1 - \frac{\gamma_{\min}}{b} \right)} \left(1 - \left(\frac{1-\epsilon}{1+\epsilon} \right)^{i+1} \left(1 - \frac{\gamma_{\min}}{b} \right)^{i+1} \right) \cdot \text{OPT}. \quad (17.23)$$

As $|s'|_1 = |s|_1 + 1 \leq i + 1$, s' will enter into P ; otherwise, s' must be dominated by one solution in P in line 6 of Algorithm 14.2, and this implies that J_{\max} is larger than i , contradicting with the assumption $J_{\max} = i$. After including s' , $J_{\max} \geq i + 1$. Let P_{\max} denote the largest size of P during the running of POSS. Thus, J_{\max} can increase by at least 1 in one iteration with probability at least $(1/P_{\max}) \cdot (1/n)(1-1/n)^{n-1} \geq 1/(enP_{\max})$, where $1/P_{\max}$ is a lower bound of the probability of selecting s in line 4 of Algorithm 14.2 and $(1/n)(1-1/n)^{n-1}$ is the probability of flipping only a specific bit of s in line 5. To increase J_{\max} , it requires at most enP_{\max} expected number of iterations. Thus, after $b \cdot enP_{\max}$ expected number of iterations, J_{\max} must have reached b . As the analysis in the proof of Theorem 14.1, $P_{\max} \leq 2b$, implying that the expected number $\mathbb{E}[T]$ of iterations for finding the desired solution is at most $2eb^2n$. \square

The approximation ratio of POSS under additive noise is shown in Theorem 17.4, the proof of which is similar to that of Theorem 17.3 except that Eq. (17.2) is used rather than Eq. (17.1).

Theorem 17.4. *For subset selection under additive noise, POSS with $\mathbb{E}[T] \leq 2eb^2n$ finds a solution s with $|s|_1 \leq b$ and*

$$f(s) \geq \left(1 - \left(1 - \frac{\gamma_{\min}}{b} \right)^b \right) \cdot \left(\text{OPT} - \frac{2be}{\gamma_{\min}} \right) - \left(1 - \frac{\gamma_{\min}}{b} \right)^b \epsilon. \quad (17.24)$$

By comparing Theorems 17.1 and 17.3, we find that the approximation bounds of POSS and the greedy algorithm under multiplicative noise are nearly the same. Particularly, for the submodular case where $\forall s, k : \gamma_{s,k} = 1$, they are exactly the same. Under additive noise, their approximation bounds in Theorems 17.2 and 17.4 are also nearly the same, because the additional term $(1 - \gamma_{\min}/b)^b \epsilon$ in Theorem 17.4 can almost be omitted compared with other terms.

To further examine the performances of the greedy algorithm and POSS, we compare them on two maximum coverage examples with noise. As introduced in Section 14.1.1, maximum coverage in Definition 14.3 is a classic subset selection problem with monotone submodular objective functions. Given a family of sets that cover a universe of elements, the goal is to select at most b sets whose union is maximum. As for Examples 17.1 and 17.2, the greedy algorithm easily finds an optimal solution if without noise, but can only guarantee nearly a $(2/b)$ and $(3/4)$ -approximation under noise, respectively. We prove in Theorems 17.5 and 17.6 that POSS can avoid the misleading search direction due to noise through multi-bit search and backward search, respectively, and find an optimal solution. Note that the greedy algorithm can perform only single-bit forward search.

Example 17.1. [Hassidim and Singer, 2017] The parameters of maximum coverage in Definition 14.3 are set as: as shown in Figure 17.1(a), V contains $n = 2l$ subsets $\{S_1, \dots, S_{2l}\}$, where $\forall i \leq l$, S_i covers the same two elements, and $\forall i > l$, S_i covers one unique element; the budget satisfies $2 < b \leq l$. The objective evaluation is exact except that $\forall \emptyset \subset S \subseteq \{S_1, \dots, S_l\}, i > l : f^n(S) = 2 + \delta$ and $f^n(S \cup \{S_i\}) = 2$, where $0 < \delta < 1$.

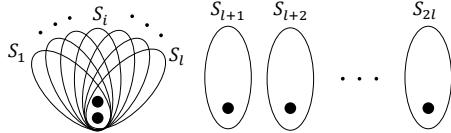
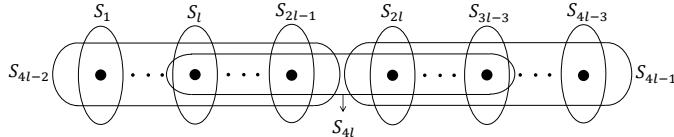
Theorem 17.5. *For Example 17.1, POSS with $\mathbb{E}[T] = O(bn \log n)$ finds an optimal solution, whereas the greedy algorithm cannot.*

Example 17.2. The parameters of maximum coverage in Definition 14.3 are set as: as shown in Figure 17.1(b), V contains $n = 4l$ subsets $\{S_1, \dots, S_{4l}\}$, where $\forall i \leq 4l - 3 : |S_i| = 1$, $|S_{4l-2}| = 2l - 1$, and $|S_{4l-1}| = |S_{4l}| = 2l - 2$; the budget $b = 2$. The objective evaluation is exact except $f^n(\{S_{4l}\}) = 2l$.

Theorem 17.6. *For Example 17.2, POSS with $\mathbb{E}[T] = O(n)$ finds the optimal solution $\{S_{4l-2}, S_{4l-1}\}$, whereas the greedy algorithm cannot.*

17.2 The PONSS Algorithm

POSS compares two solutions based on the domination relation in Definition 1.2. This may be less robust against noise, because a worse solution can appear to have a better f^n value and then survive to replace a truly better solution. Note that the isolation function I in POSS is assumed to

(a) An example with $n = 2l$ subsets [Hassidim and Singer, 2017](b) An example with $n = 4l$ subsets**Figure 17.1.** Two examples of the maximum coverage problem.

be a constant function, as it is generally good. Inspired by the noise handling strategy, i.e., threshold selection, analyzed in Chapter 10, we modify POSS by replacing domination with θ -domination, where s is better than s' if $f^n(s)$ is larger than $f^n(s')$ by at least a threshold. By θ -domination, solutions with similar f^n values will be kept in P rather than only one with the best f^n value is kept; thus, the risk of removing a good solution is reduced. This modified Pareto Optimization algorithm for Noisy Subset Selection (PONSS) is presented in Algorithm 17.1.

For multiplicative noise, we use the multiplicative θ -domination relation in Definition 17.1. In other words, $s \succeq_\theta s'$ if $f^n(s) \geq ((1 + \theta)/(1 - \theta)) \cdot f^n(s')$ and $|s|_1 \leq |s'|_1$.

Definition 17.1 (Multiplicative θ -Domination). For two solutions s and s' ,

- s weakly dominates s' , denoted as $s \succeq_\theta s'$, if

$$f_1(s) \geq \frac{1 + \theta}{1 - \theta} \cdot f_1(s') \wedge f_2(s) \geq f_2(s'); \quad (17.25)$$

- s dominates s' , denoted as $s \succ_\theta s'$, if

$$s \succeq_\theta s' \wedge \left(f_1(s) > \frac{1 + \theta}{1 - \theta} \cdot f_1(s') \vee f_2(s) > f_2(s') \right). \quad (17.26)$$

For additive noise, we use the additive θ -domination relation in Definition 17.2. In other words, $s \succeq_\theta s'$ if $f^n(s) \geq f^n(s') + 2\theta$ and $|s|_1 \leq |s'|_1$.

Definition 17.2 (Additive θ -Domination). For two solutions s and s' ,

- s weakly dominates s' , denoted as $s \succeq_\theta s'$, if

$$f_1(s) \geq f_1(s') + 2\theta \wedge f_2(s) \geq f_2(s'); \quad (17.27)$$

- s dominates s' , denoted as $s \succ_\theta s'$, if

$$s \succeq_\theta s' \wedge (f_1(s) > f_1(s') + 2\theta \vee f_2(s) > f_2(s')). \quad (17.28)$$

Algorithm 17.1 PONSS Algorithm

Input: $V = \{v_1, v_2, \dots, v_n\}$; noisy objective function $f^n : \{0, 1\}^n \rightarrow \mathbb{R}$; budget $b \in [n]$

Parameter: $T; \theta; l$

Output: solution $s \in \{0, 1\}^n$ with $|s|_1 \leq b$

Process:

- 1: let $s = 0^n$ and $P = \{s\}$;
- 2: let $t = 0$;
- 3: **while** $t < T$ **do**
- 4: select a solution s from P uniformly at random;
- 5: apply bit-wise mutation on s to generate s' ;
- 6: **if** $\nexists z \in P$ such that $z \succ_\theta s'$ **then**
- 7: $P = (P \setminus \{z \in P \mid s' \succeq_\theta z\}) \cup \{s'\}$;
- 8: $Q = \{z \in P \mid |z|_1 = |s'|_1\}$;
- 9: **if** $|Q| = l + 1$ **then**
- 10: $P = P \setminus Q$ and let $j = 0$;
- 11: **while** $j < l$ **do**
- 12: select two solutions z_1, z_2 from Q uniformly at random;
- 13: evaluate $f^n(z_1), f^n(z_2)$, and let $\hat{z} = \arg \max_{z \in \{z_1, z_2\}} f^n(z)$;
- 14: $P = P \cup \{\hat{z}\}, Q = Q \setminus \{\hat{z}\}$, and $j = j + 1$
- 15: **end while**
- 16: **end if**
- 17: **end if**
- 18: $t = t + 1$
- 19: **end while**
- 20: **return** $\arg \max_{s \in P, |s|_1 \leq b} f_1(s)$

Using θ -domination may make the size of the population P very large, and then reduce the efficiency of PONSS. We further introduce a parameter l to limit the number of solutions in P for each possible subset size. In other words, if the number of solutions with the same size in P exceeds l , one of them will be deleted. As shown in lines 8-16 of Algorithm 17.1, the better one of two solutions randomly selected from Q is kept; this process is repeated for l times, and the remaining solution in Q is deleted.

17.3 Theoretical Analysis

For the analysis of PONSS, we consider random noise, i.e., $f^n(s)$ is a random variable, and assume that the probability of $f^n(s) > f^n(s')$ is no less than $0.5 + \delta$ if $f(s) > f(s')$, i.e.,

$$P(f^n(s) > f^n(s')) \geq 0.5 + \delta \quad \text{if } f(s) > f(s'), \quad (17.29)$$

where $\delta \in [0, 0.5]$. Note that the value of δ depends on the concrete noise distribution. This assumption is satisfied in many noisy settings, e.g., the

noise distribution is i.i.d. for each s . If $f^n(s) = f(s) + \xi(s)$, where the noise $\xi(s)$ is drawn independently from the same distribution for each s , we have, for two solutions s and s' with $f(s) > f(s')$,

$$\begin{aligned} P(f^n(s) > f^n(s')) &= P(f(s) + \xi(s) > f(s') + \xi(s')) \\ &\geq P(\xi(s) \geq \xi(s')) \end{aligned} \quad (17.30)$$

$$\geq 0.5, \quad (17.31)$$

where Eq. (17.30) holds by the condition that $f(s) > f(s')$, and Eq. (17.31) holds by $P(\xi(s) \geq \xi(s')) + P(\xi(s) \leq \xi(s')) \geq 1$ and $P(\xi(s) \geq \xi(s')) = P(\xi(s) \leq \xi(s'))$ due to the fact that $\xi(s)$ and $\xi(s')$ are from the same distribution. If $f^n(s) = f(s) \cdot \xi(s)$, Eq. (17.29) holds similarly. Note that for comparing two solutions selected from Q in line 13 of PONSS, their noisy objective f^n values are re-evaluated independently, i.e., each evaluation is a new independent random draw from the noise distribution.

For the multiplicative noise model, Theorem 17.7 shows the approximation ratio of PONSS under the assumption Eq. (17.29), better than that of POSS under general multiplicative noise in Theorem 17.3, because

$$\begin{aligned} \frac{1 - \left(\frac{1-\epsilon}{1+\epsilon}\right)^b \left(1 - \frac{\gamma_{\min}}{b}\right)^b}{1 - \frac{1-\epsilon}{1+\epsilon} \left(1 - \frac{\gamma_{\min}}{b}\right)} &= \sum_{i=0}^{b-1} \left(\frac{1-\epsilon}{1+\epsilon} \left(1 - \frac{\gamma_{\min}}{b}\right) \right)^i \\ &\leq \sum_{i=0}^{b-1} \left(1 - \frac{\gamma_{\min}}{b}\right)^i = \frac{1 - \left(1 - \frac{\gamma_{\min}}{b}\right)^b}{\frac{\gamma_{\min}}{b}}. \end{aligned} \quad (17.32)$$

Particularly for the submodular case where $\gamma_{\min} = 1$, PONSS under the assumption Eq. (17.29) can achieve a constant approximation ratio even when ϵ is a constant, whereas the greedy algorithm and POSS under general multiplicative noise guarantee only a $\Theta(1/k)$ -approximation ratio. Note that when δ is a constant, the approximation guarantee of PONSS can hold with a constant probability by using a polynomially large l , and thus, the number of iterations of PONSS is polynomial in expectation.

Theorem 17.7. *For subset selection under multiplicative noise with the assumption Eq. (17.29), with probability at least $(1/2)(1 - (12nb^2 \log 2b)/l^{2\delta})$, PONSS with $\theta \geq \epsilon$ and $T = 2elnb^2 \log 2b$ finds a solution s with $|s|_1 \leq b$ and*

$$f(s) \geq \frac{1-\epsilon}{1+\epsilon} \left(1 - \left(1 - \frac{\gamma_{\min}}{b}\right)^b\right) \cdot \text{OPT}. \quad (17.33)$$

Proof. Let J_{\max} denote the maximum value of $j \in [b]$ such that $\exists s \in P$ with $|s|_1 \leq j$ and $f(s) \geq (1 - (1 - \gamma_{\min}/b)^j) \cdot \text{OPT}$. $J_{\max} = b$ implies that $\exists s^* \in P$ satisfying that $|s^*|_1 \leq b$ and $f(s^*) \geq (1 - (1 - \gamma_{\min}/b)^b) \cdot \text{OPT}$. Because the final solution s selected from P in line 20 of Algorithm 17.1 has the largest f^n value, we have

$$f(\mathbf{s}) \geq \frac{1}{1+\epsilon} f^n(\mathbf{s}) \geq \frac{1}{1+\epsilon} f^n(\mathbf{s}^*) \geq \frac{1-\epsilon}{1+\epsilon} f(\mathbf{s}^*); \quad (17.34)$$

that is, the desired approximation bound is reached. Thus, we only need to analyze the probability of $J_{\max} = b$ after running $2elnb^2 \log 2b$ iterations.

Assume that in the running of PONSS, one solution with the best f value in Q is always kept after each implementation of lines 9-16. Next we show that J_{\max} can reach b with probability at least $1/2$ after running $2elnb^2 \log 2b$ iterations. J_{\max} is initially 0 as PONSS starts from 0^n . Assume that currently $J_{\max} = i < b$. Let \mathbf{s} denote a corresponding solution, i.e., $|\mathbf{s}|_1 \leq i$ and $f(\mathbf{s}) \geq (1 - (1 - \gamma_{\min}/b)^i) \cdot \text{OPT}$. First, J_{\max} will not decrease. It holds if \mathbf{s} is not deleted. For deleting \mathbf{s} , there are two possible cases. If \mathbf{s} is deleted in line 7, the newly included solution $\mathbf{s}' \succeq_{\theta} \mathbf{s}$, implying that $|\mathbf{s}'|_1 \leq |\mathbf{s}|_1 \leq i$ and

$$f(\mathbf{s}') \geq \frac{1}{1+\epsilon} f^n(\mathbf{s}') \geq \frac{1}{1+\epsilon} \cdot \frac{1+\theta}{1-\theta} f^n(\mathbf{s}) \geq \frac{1}{1+\epsilon} \cdot \frac{1+\epsilon}{1-\epsilon} f^n(\mathbf{s}) \geq f(\mathbf{s}), \quad (17.35)$$

where the third inequality holds by $\theta \geq \epsilon$. If \mathbf{s} is deleted in lines 9-16, there must exist one solution \mathbf{z}^* in P with $|\mathbf{z}^*|_1 = |\mathbf{s}|_1$ and $f(\mathbf{z}^*) \geq f(\mathbf{s})$, because we assume that one solution with the best f value in Q is kept. Second, J_{\max} can increase in each iteration with some probability. From Lemma 14.1, a new solution \mathbf{s}' can be generated by flipping one specific 0-bit of \mathbf{s} , i.e., adding a specific item, such that $|\mathbf{s}'|_1 = |\mathbf{s}|_1 + 1 \leq i + 1$ and

$$f(\mathbf{s}') \geq \left(1 - \left(1 - \frac{\gamma_{\min}}{b}\right)^{i+1}\right) \cdot \text{OPT}. \quad (17.36)$$

Note that Eq. (17.36) can be derived using the same analysis as Eq. (14.21). \mathbf{s}' will enter into P ; otherwise, there must exist one solution in P dominating \mathbf{s}' in line 6 of Algorithm 17.1, and this implies that J_{\max} is larger than i , contradicting with the assumption $J_{\max} = i$. After including \mathbf{s}' , $J_{\max} \geq i + 1$. As P contains at most l solutions for each possible size $\{0, \dots, 2b - 1\}$, we have $|P| \leq 2bl$. Thus, J_{\max} can increase by at least 1 in one iteration with probability at least $(1/|P|) \cdot (1/n)(1 - 1/n)^{n-1} \geq 1/(2elnb)$, where $1/|P|$ is the probability of selecting \mathbf{s} in line 4 and $(1/n)(1 - 1/n)^{n-1}$ is the probability of flipping only a specific bit of \mathbf{s} in line 5. We divide the $2elnb^2 \log 2b$ iterations into b phases with equal length. For reaching $J_{\max} = b$, it is sufficient that J_{\max} increases at least once in each phase. Thus,

$$P(J_{\max} = b) \geq \left(1 - \left(1 - \frac{1}{2elnb}\right)^{2elnb \log 2b}\right)^b \geq \left(1 - \frac{1}{2b}\right)^b \geq \frac{1}{2}. \quad (17.37)$$

Next we examine our assumption that during the running of $2elnb^2 \log 2b$ iterations, when implementing lines 9-16 of Algorithm 17.1, one solution with the best f value in Q is always kept. Let $R = \{\mathbf{z}^* = \arg \max_{\mathbf{z} \in Q} f(\mathbf{z})\}$. If $|R| > 1$, it holds as only one solution from Q is deleted. If $|R| = 1$, deleting

the solution z^* with the best f value implies that z^* is never included into P in implementing lines 12-14, which are repeated for l iterations. In the j -th iteration where $0 \leq j \leq l - 1$, $|Q| = l + 1 - j$. Under the condition that z^* is never included into P from the 0-th to the $(j - 1)$ -th iteration, the probability of selecting z^* in line 12 is $(l - j)/\binom{l+1-j}{2} = 2/(l + 1 - j)$. By Eq. (17.29), $f^n(z^*)$ is better in the comparison of line 13 with probability at least $0.5 + \delta$. Thus, the probability of not including z^* into P in the j -th iteration is at most $1 - (2/(l + 1 - j)) \cdot (0.5 + \delta)$, implying that the probability of deleting the solution with the best f value in Q when implementing lines 9-16 is at most $\prod_{j=0}^{l-1} (1 - (1 + 2\delta)/(l + 1 - j))$. Taking the logarithm, we have

$$\sum_{j=0}^{l-1} \log \left(\frac{l - j - 2\delta}{l + 1 - j} \right) = \sum_{j=1}^l \log \left(\frac{j - 2\delta}{j + 1} \right) \leq \int_1^{l+1} \log \left(\frac{j - 2\delta}{j + 1} \right) dj \quad (17.38)$$

$$= \log \left(\frac{(l + 1 - 2\delta)^{l+1-2\delta}}{(l + 2)^{l+2}} \right) - \log \left(\frac{(1 - 2\delta)^{1-2\delta}}{2^2} \right), \quad (17.39)$$

where Eq. (17.38) holds because $\log((j - 2\delta)/(j + 1))$ is increasing with j , and Eq. (17.39) holds because the derivative of $\log((j - 2\delta)^{j-2\delta}/(j + 1)^{j+1})$ with respect to j is $\log((j - 2\delta)/(j + 1))$. Thus, we have

$$\begin{aligned} \prod_{j=0}^{l-1} \left(1 - \frac{1 + 2\delta}{l + 1 - j} \right) &\leq \left(\frac{l + 1 - 2\delta}{l + 2} \right)^{l+2} \cdot \frac{1}{(l + 1 - 2\delta)^{1+2\delta}} \cdot \frac{4}{(1 - 2\delta)^{1-2\delta}} \\ &\leq \frac{4}{e^{1-1/e} l^{1+2\delta}}, \end{aligned} \quad (17.40)$$

where Eq. (17.40) holds by $0 < 1 - 2\delta \leq 1$ and $(1 - 2\delta)^{1-2\delta} \geq e^{-1/e}$. By the union bound, our assumption holds with probability at least $1 - (12nb^2 \log 2b)/l^{2\delta}$. Thus, the theorem holds. \square

By applying Eq. (17.2) and additive θ -domination to the proof procedure of Theorem 17.7, we prove the approximation ratio of PONSS under additive noise with the assumption Eq. (17.29), as shown in Theorem 17.8. Compared with the approximation ratio of POSS under general additive noise in Theorem 17.4, PONSS achieves a better one. This can be verified because $(1 - (1 - \gamma_{\min}/b)^b)(2b\epsilon/\gamma_{\min}) \geq 2\epsilon$, where the inequality holds by $\gamma_{\min} \in [0, 1]$.

Theorem 17.8. *For subset selection under additive noise with the assumption Eq. (17.29), with probability at least $(1/2)(1 - (12nb^2 \log 2b)/l^{2\delta})$, PONSS with $\theta \geq \epsilon$ and $T = 2elnb^2 \log 2b$ finds a solution s with $|s|_1 \leq b$ and*

$$f(s) \geq \left(1 - \left(1 - \frac{\gamma_{\min}}{b} \right)^b \right) \cdot \text{OPT} - 2\epsilon. \quad (17.41)$$

17.4 Empirical Study

In this section, we empirically compare the performance of the greedy algorithm, POSS and PONSS. We conduct experiments on two typical subset selection problems: influence maximization and sparse regression, where the former has a submodular objective function and the latter has a non-submodular one. The number T of iterations in POSS is set to $2eb^2n$ suggested by Theorem 17.3. For PONSS, l is set to b , and θ is set to 1, which is no smaller than ϵ . Note that POSS requires one objective evaluation for the newly generated solution s' in each iteration, whereas PONSS requires 1 or $(1 + 2l)$ evaluations, depending on whether the condition in line 9 of Algorithm 17.1 is satisfied. For the fairness of comparison, PONSS is terminated until the total number of evaluations reaches that of POSS, i.e., $2eb^2n$. We test the budget b from 3 to 8. During the running of each algorithm, only a noisy objective f^n value can be obtained; whereas for the final output solution, its accurately estimated f value by an expensive evaluation is reported for the assessment of the algorithms. As POSS and PONSS are randomized algorithms and the behavior of the greedy algorithm is also randomized under random noise, we repeat the running for 10 times independently and report the average estimated f values.

As introduced in Section 14.1.2, influence maximization is to find a subset S of social network users with $|S| \leq b$ such that the expected number of users activated by propagating from S , called influence spread, is maximized. We use two real-world data sets: *ego-Facebook* and *weibo*, in Table 15.1. For estimating the objective, influence spread, we simulate the diffusion process for 10 times independently and use the average as an estimation. For the final output solutions of the algorithms, we average over 10,000 times for accurate estimation.

From the left subfigure on each data set in Figure 17.2, it can be seen that POSS is better than the greedy algorithm, and PONSS performs the best. By selecting the greedy algorithm for the baseline, we plot in the right subfigures the curve of influence spread over running time for PONSS and POSS with $b = 5$. Note that the x -axis is in bn , i.e., the running time order of the greedy algorithm. It can be seen that PONSS quickly reaches better performance, implying that PONSS can be efficient in practice.

As introduced in Section 14.1.5, sparse regression is to find a subset S of at most b observation variables to best approximate a predictor variable z by linear regression. The problem is formulated as maximizing the squared multiple correlation $R_{z,S}^2$ in Eq. (14.13). We use two data sets: *protein* and *YearPredictionMSD*.[†] For estimating R^2 in the optimization process, a random sample of 1,000 instances is used. For the final output solutions, the whole data set is used for accurate estimation. The results are plotted in Figure 17.3. The performance of the three algorithms is similar to that ob-

[†]<http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>.

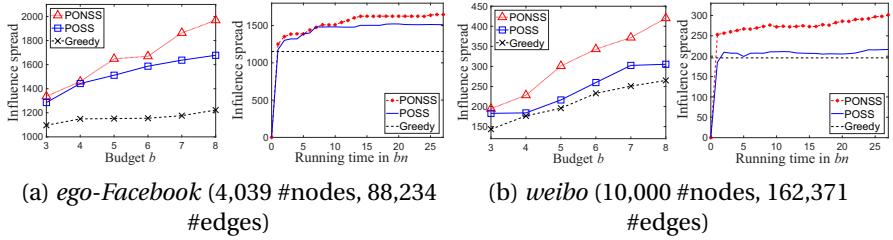


Figure 17.2. Influence maximization (influence spread: the larger, the better). The right subfigure on each data set: influence spread vs. running time of PONSS, POSS and the greedy algorithm for the budget $b = 5$.

served for influence maximization, except the loss of POSS over the greedy algorithm on *YearPredictionMSD* with $b = 8$.

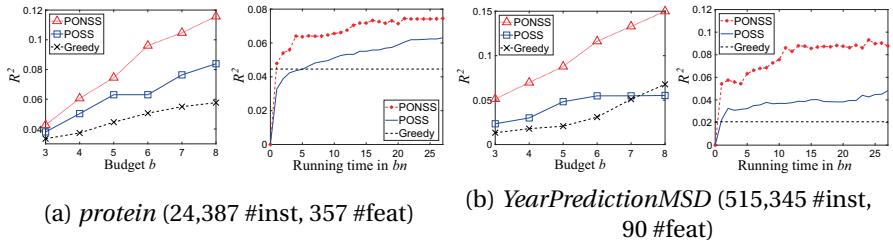


Figure 17.3. Sparse regression (R^2 : the larger, the better). The right subfigure on each data set: R^2 vs. running time of PONSS, POSS and the greedy algorithm for the budget $b = 5$.

For both tasks, we test PONSS with $\theta = \{0.1, 0.2, \dots, 1\}$. The results are shown in Figure 17.4. It can be observed that PONSS is always better than POSS and the greedy algorithm, implying that the performance of PONSS is not sensitive to the value of θ .

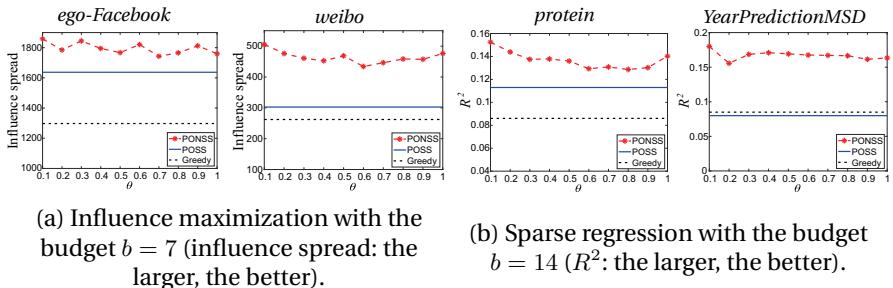


Figure 17.4. Comparison between PONSS, POSS and the greedy algorithm, where the parameter θ of PONSS is set to $0.1, 0.2, \dots, 1$, respectively.

17.5 Summary

In this chapter, we study the subset selection problem with monotone objective functions under multiplicative and additive noise. We first show that the greedy algorithm and POSS, two powerful algorithms for noise-free subset selection, achieve nearly the same approximation guarantee under noise. Next, we present the PONSS algorithm, which can achieve a better approximation ratio under some assumption such as i.i.d. noise distribution. Empirical results on influence maximization and sparse regression verify the excellent performance of PONSS.



Subset Selection: Acceleration

In chapter 14, we have introduced the POSS algorithm for subset selection. POSS requires calling $2eb^2n$ number of objective function evaluations to achieve a high-quality solution, which can be impractical when b and n are large. Moreover, POSS is a sequential algorithm which cannot be readily parallelized; this hinders the exploitation of modern computer facilities when applying to large-scale tasks.

In this chapter, we present a parallel version of POSS, called PPOSS [Qian et al., 2016a]. Rather than generating one solution at a time as in POSS, PPOSS generates as many solutions as the number of processors at a time, and can be easily parallelized. More importantly, on subset selection with monotone objective functions, we prove the following result while preserving the solution quality: When the number of processors is limited, i.e., less than the number n of items, the running time of PPOSS can be reduced almost linearly with respect to the number of processors; with increasing number of processors, the running time can be continuously reduced, eventually to a constant. Experimental results on the application of sparse regression verify our theoretical analysis, and suggest that the lock-free implementation of PPOSS is more efficient, while introducing little loss in solution quality.

The rest of this chapter is organized as follows. Sections 18.1 to 18.3 present the PPOSS algorithm, theoretical analysis and empirical study, respectively. Section 18.4 concludes this chapter.

18.1 The PPOSS Algorithm

The POSS algorithm presented in Chapter 14 reformulates the subset selection problem Eq. (14.5) into a bi-objective maximization problem Eq. (14.15), and solves this bi-objective problem by a simple MOEA. As introduced in Section 14.2, POSS generates a new solution by mutation and uses it to update the population in each iteration. In POSS, an isolation function I is

introduced to determine if two solutions can be compared. As a constant isolation function is generally useful, I is set to a constant function here, which can be ignored because every solution has the same I value.

For subset selection with monotone objective functions, POSS, using at most $2eb^2n$ expected number of iterations, has been proved to find a solution with $|S| \leq b$ and $f(S) \geq (1 - e^{-\gamma}) \cdot \text{OPT}$, as shown in Theorem 14.1. For submodular objective functions, $\gamma = 1$ and the obtained approximation guarantee $(1 - e^{-1})$ is optimal in general [Nemhauser and Wolsey, 1978]. For a representative application, sparse regression, where the objective function can be non-submodular, the approximation guarantee $(1 - e^{-\gamma})$ obtained by POSS is the best known one [Das and Kempe, 2011].

Algorithm 18.1 PPOSS Algorithm

Input: $V = \{v_1, v_2, \dots, v_n\}$; objective function $f : \{0, 1\}^n \rightarrow \mathbb{R}$; budget $b \in [n]$

Parameter: the number T of iterations; the number m of processors

Output: solution $s \in \{0, 1\}^n$ with $|s|_1 \leq b$

Process:

```

1: let  $s = 0^n$  and  $P = \{s\}$ ;
2: let  $t = 0$ ;
3: while  $t < T$  do
4:   select a solution  $s$  from  $P$  uniformly at random;
5:   begin parallel on  $m$  processors
6:     apply bit-wise mutation on  $s$  to generate  $s'_i$ ;
7:     evaluate  $f_1(s'_i)$  and  $f_2(s'_i)$ 
8:   end parallel
9:   for each  $s'_i$ 
10:    if  $\nexists z \in P$  such that  $z \succ s'_i$  then
11:       $Q = \{z \in P \mid s'_i \succeq z\}$ ;
12:       $P = (P \setminus Q) \cup \{s'_i\}$ 
13:    end if
14:   end for
15:    $t = t + 1$ 
16: end while
17: return  $\arg \max_{s \in P, |s|_1 \leq b} f_1(s)$ 

```

Parallel POSS (PPOSS) is shown in Algorithm 18.1, which simply modifies POSS in Algorithm 14.2 to generate as many solutions as the number of processors in an iteration. In each iteration of PPOSS, it first randomly selects a solution s from the current population P in line 4, and then implements lines 6-7 in parallel. At each processor, it independently generates a new solution s' from s by mutation in line 6, and evaluates both two objective values in line 7. Note that s'_i denotes s' at the i -th processor. After the procedure of parallelization, those newly generated solutions are used to update the population P in lines 9-14, making P always maintain non-dominated solutions generated-so-far. Compared with POSS in Algo-

rithm 14.2, PPOSS generates multiple new solutions in parallel in each iteration rather than generating only one new solution. The procedures of one iteration for POSS and PPOSS are illustrated in Figure 18.1. When the number m of processors is 1, PPOSS is just POSS.

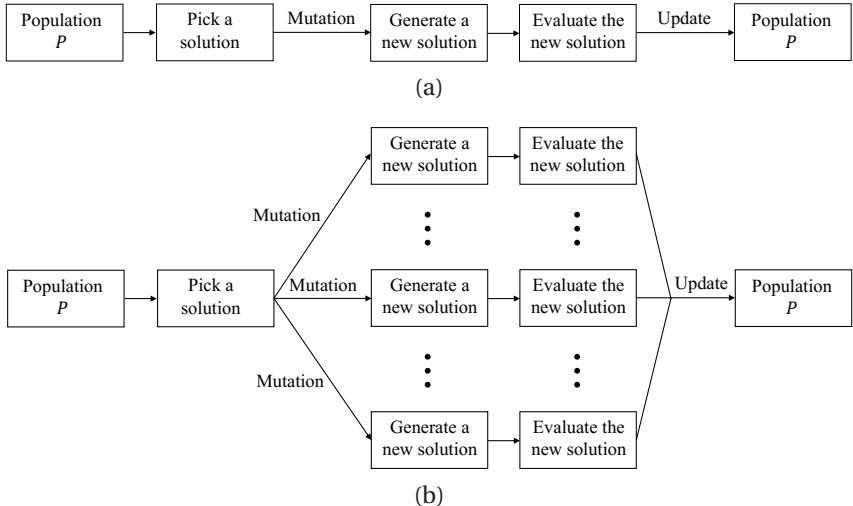


Figure 18.1. Illustration of one iteration of (a) POSS and (b) PPOSS.

18.2 Theoretical Analysis

In this section, we theoretically analyze the performance of PPOSS on subset selection with monotone objective functions. We prove the approximation bound of PPOSS in Theorem 18.1. To find a solution with the approximation guarantee $1 - e^{-\gamma_{\min}}$, PPOSS achieves linear speedup in the number T of iterations when the number m of processors is asymptotically smaller than the number n of items; the required T can reduce to $O(1)$ when m is sufficiently large. The proof requires Lemma 14.1 which states that $\forall S \subseteq V$ there always exists another item whose inclusion can improve f by at least a quantity proportional to the current distance to the optimal function value.

Theorem 18.1. *For subset selection with monotone objective functions, the expected number of iterations until PPOSS finds a solution s with $|s|_1 \leq b$ and $f(s) \geq (1 - e^{-\gamma_{\min}}) \cdot \text{OPT}$, where $\gamma_{\min} = \min_{s:|s|_1=b-1} \gamma_{s,b}$, is*

- (1) *if $m = o(n)$, then $\mathbb{E}[T] \leq 2eb^2n/m$;*
- (2) *if $m = \Omega(n^i)$ for $1 \leq i \leq b$, then $\mathbb{E}[T] = O(b^2/i)$;*
- (3) *if $m = \Omega(n^{\min\{3b-1,n\}})$, then $\mathbb{E}[T] = O(1)$.*

Proof. The theorem is proved by analyzing the increase of a quantity J_{\max} , which is the maximum value of $j \in [b]$ such that $\exists s \in P$ with $|s|_1 \leq j$ and $f(s) \geq (1 - (1 - \gamma_{\min}/b)^j) \cdot \text{OPT}$. In other words, $J_{\max} = \max\{j \in [b] \mid \exists s \in P : |s|_1 \leq j \wedge f(s) \geq (1 - (1 - \gamma_{\min}/b)^j) \cdot \text{OPT}\}$. As PPOSS starts from the solution 0^n in line 1 of Algorithm 18.1, the initial value of J_{\max} is 0. $J_{\max} = b$ implies that we have found a solution s in P satisfying that $|s|_1 \leq b$ and $f(s) \geq (1 - (1 - \gamma_{\min}/b)^b) \cdot \text{OPT} \geq (1 - e^{-\gamma_{\min}}) \cdot \text{OPT}$. Thus, we just need to analyze the expected number of iterations until $J_{\max} = b$.

Assume that $J_{\max} = i < b$. Let s be a corresponding solution with the value i , i.e., $|s|_1 \leq i$ and $f(s) \geq (1 - (1 - \gamma_{\min}/b)^i) \cdot \text{OPT}$. If s is kept in P , J_{\max} will not decrease. If s is removed, there must exist one newly generated solution s'_i weakly dominating s , and s'_i will enter into P in lines 11-12 of Algorithm 18.1. Considering $s'_i \succeq s$, we have $|s'_i|_1 \leq |s|_1 \leq i \wedge f(s'_i) \geq f(s) \geq (1 - (1 - \gamma_{\min}/b)^i) \cdot \text{OPT}$. Thus, $J_{\max} \geq i$, i.e., J_{\max} does not decrease.

Next we show that J_{\max} can always increase by at least $l \in [b - i]$ in one iteration with some probability. According to Lemma 14.1, flipping one specific 0-bit of s can generate a new solution s' , satisfying $f(s') - f(s) \geq (\gamma_{s,b}/b)(\text{OPT} - f(s))$. As the analysis of Eq. (14.21), we have

$$f(s') \geq \left(1 - \left(1 - \frac{\gamma_{\min}}{b}\right)^{i+1}\right) \cdot \text{OPT}. \quad (18.1)$$

Implementing this step for l times sequentially generates a solution s' with

$$f(s') \geq \left(1 - \left(1 - \frac{\gamma_{\min}}{b}\right)^{i+l}\right) \cdot \text{OPT}. \quad (18.2)$$

This also implies that such a solution can be found by flipping l specific 0-bits of s simultaneously in one iteration. Note that $|s'|_1 = |s|_1 + l \leq i + l$. Once generated, s' cannot be dominated by any solution in the current P ; otherwise, J_{\max} is larger than i , contradicting with the assumption $J_{\max} = i$. Thus, s' will enter into P in line 12, making $J_{\max} \geq i + l$. We then only need to analyze the probability of generating s' in one iteration. It is sufficient to select s in line 4 and then flip those l specific 0-bits in at least one processor. Let P_{\max} denote the largest size of P during the running of PPOSS. The probability of selecting s in line 4 is at least $1/P_{\max}$ due to uniform selection. The probability of flipping l specific bits of s while keeping the other bits unchanged in line 6 is $(1/n^l)(1 - 1/n)^{n-l}$. Because line 6 is implemented independently by each processor, the probability of generating s' is at least

$$\frac{1}{P_{\max}} \cdot \left(1 - \left(1 - \frac{1}{n^l} \left(1 - \frac{1}{n}\right)^{n-l}\right)^m\right) \geq \frac{1}{P_{\max}} \cdot \left(1 - \left(1 - \frac{1}{en^l}\right)^m\right) \quad (18.3)$$

$$\geq \frac{1}{P_{\max}} \cdot \left(1 - e^{-\frac{m}{en^l}}\right), \quad (18.4)$$

where Eq. (18.3) holds by $(1 - 1/n)^{n-l} \geq 1/e$ for $l \geq 1$, and Eq. (18.4) holds by $(1 - 1/en^l)^{en^l} \leq 1/e$.

Thus, we have shown that (1) J_{\max} cannot decrease; (2) J_{\max} can increase by at least l in one iteration with probability at least $(1/P_{\max}) \cdot (1 - e^{-m/(en^l)})$. According to these two points, the expected number of iterations for increasing J_{\max} by at least l , called one *successful step*, is at most $P_{\max}/(1 - e^{-m/(en^l)})$. Because $\lceil b/l \rceil$ successful steps are sufficient to make $J_{\max} = b$, the total expected number of iterations is

$$\mathbb{E}[T] \leq \lceil b/l \rceil \cdot \frac{P_{\max}}{1 - e^{-\frac{m}{en^l}}}. \quad (18.5)$$

From lines 9-14 of Algorithm 18.1, any two solutions in the population P must be incomparable. As in the proof of Theorem 14.1, we can show that $P_{\max} \leq 2b$, leading to

$$\mathbb{E}[T] \leq \frac{2b\lceil b/l \rceil}{1 - e^{-\frac{m}{en^l}}}, \quad (18.6)$$

where $1 \leq l \leq b$. If $m = o(n)$, let $l = 1$, and then $e^{-m/(en)} = 1 - m/(en) + O((m/n)^2) \approx 1 - m/(en)$. Thus, $\mathbb{E}[T] \leq 2eb^2n/m$. If $m = \Omega(n^i)$ for $1 \leq i \leq b$, let $l = i$, and then $1 - e^{-m/(en^i)} = \Theta(1)$. Thus, we have $\mathbb{E}[T] = O(b^2/i)$.

We finally consider case (3) by analyzing the probability P_{opt} of generating an optimal solution in one iteration. Considering that any solution s in P has $|s|_1 \leq 2b-1$ and an optimal solution contains at most b number of 1-bits, the Hamming distance between any solution in P and an optimal solution is at most $\min\{3b-1, n\}$. Regardless of the selected solution in line 4, the probability of generating an optimal solution in one iteration at each processor is at least $(1/n^{\min\{3b-1, n\}})(1 - 1/n)^{n-\min\{3b-1, n\}} \geq 1/(en^{\min\{3b-1, n\}})$. As it is sufficient to generate an optimal solution in at least one processor, the probability P_{opt} is at least $1 - (1 - 1/(en^{\min\{3b-1, n\}}))^m = \Theta(1)$, where the equality holds by $m = \Omega(n^{\min\{3b-1, n\}})$. Thus, $\mathbb{E}[T] \leq 1/P_{opt} = O(1)$. \square

Therefore, PPOSS can achieve linear speedup in the number T of iterations for finding a solution with the approximation guarantee $1 - e^{-\gamma_{\min}}$. Compared with the approximation guarantee of POSS in Theorem 14.1, PPOSS does not degrade the performance.

The proved linear speedup of PPOSS is based on the number of iterations. Let t_{poss} and t_{pposs} denote the running time of each iteration for POSS and PPOSS, respectively. If $t_{\text{poss}} = t_{\text{pposs}}$, we can conclude that PPOSS achieves linear speedup in the running time. In the following, we will show that t_{poss} and t_{pposs} are close in expectation.

Let t_s, t_g, t_e and t_u denote the running time of selecting an archived solution s in line 4 of Algorithm 18.1, generating a new solution s' in line 6, evaluating the objective values of s' in line 7, and updating the population P in lines 10-13, respectively. Note that t_e usually depends on the number of 1-bits of s' , i.e., the number of selected items, denoted by v . For the convenience of analysis, we assume linear dependence, i.e., $t_e = c \cdot v$, where c is a

constant. The extra parallel overhead will not be considered in the analysis for simplicity.

Theorem 18.2. *The expected difference between the running time of each iteration for POSS and PPOSS is*

$$\mathbb{E}[t_{p\text{poss}} - t_{\text{poss}}] \leq (m-1) \cdot t_u + \frac{c}{2}. \quad (18.7)$$

Proof. From the procedure of Algorithms 14.2 and 18.1, we have

$$t_{\text{poss}} = t_s + t_g + t_e + t_u, \quad (18.8)$$

$$t_{p\text{poss}} = t_s + \max\{t_g^i + t_e^i \mid i \in [m]\} + m \cdot t_u, \quad (18.9)$$

where t_g^i and t_e^i denote t_g and t_e at the i -th processor, respectively. t_g is fixed, and thus, $t_{p\text{poss}} = t_s + t_g + \max\{t_e^i \mid i \in [m]\} + m \cdot t_u$, leading to

$$\begin{aligned} t_{p\text{poss}} - t_{\text{poss}} &= \max\{t_e^i \mid i \in [m]\} - t_e + (m-1)t_u \\ &= c \cdot \max\{v_{pp}^i \mid i \in [m]\} - c \cdot v_p + (m-1)t_u, \end{aligned} \quad (18.10)$$

where v_p and v_{pp} denote the number of 1-bits of the newly generated solution s' for POSS and PPOSS, respectively.

Next we analyze the expectation of $\max\{v_{pp}^i \mid i \in [m]\}$ and v_p . Let u_p and u_{pp} denote the number of 1-bits of the selected solution s in line 4 for POSS and PPOSS, respectively. Because s' is generated by flipping each bit of s with probability $1/n$, we can derive: $\mathbb{E}[v_p] = u_p + 1 - 2u_p/n$, $\mathbb{E}[v_{pp}^i] = u_{pp} + 1 - 2u_{pp}/n$, and $\text{Var}(v_p) = \text{Var}(v_{pp}^i) = 1 - 1/n$. Note that $v_{pp}^1, v_{pp}^2, \dots, v_{pp}^m$ are i.i.d. because each processor runs independently. By applying the upper bound of the expected highest order statistic [Gumbel, 1954, Hartly and David, 1954], we have

$$\mathbb{E}[\max\{v_{pp}^i \mid i \in [m]\}] \leq \mathbb{E}[v_{pp}^i] + \sqrt{\text{Var}(v_{pp}^i)} \cdot \frac{m-1}{2m-1}. \quad (18.11)$$

Because the population P contains at most one solution for each $|s|_1 = j < 2b$ and s is selected from P uniformly at random, the distributions of u_p and u_{pp} have little difference. For the convenience of analysis, assume that they are the same, denoted as p . We then have

$$\begin{aligned} &\mathbb{E}[t_{p\text{poss}} - t_{\text{poss}}] \\ &= (m-1)t_u + c \cdot \sum_j p(j) (\mathbb{E}[\max\{v_{pp}^i \mid i \in [m]\} \mid u_{pp} = j] - \mathbb{E}[v_p \mid u_p = j]) \\ &\leq (m-1)t_u + c \cdot \sum_j p(j) \cdot \sqrt{1 - \frac{1}{n}} \cdot \frac{m-1}{2m-1} \leq (m-1)t_u + \frac{c}{2}. \end{aligned} \quad (18.12)$$

□

By lines 10-13 of Algorithm 18.1, t_u is mainly the running time of comparing s'_i with the solutions in the population P . The total number of comparisons is at most $2b$, for it is the largest size of P . Compared with the objective function evaluation time $t_e = c \cdot v$, the time of comparing solutions is usually much smaller, and can even be ignored. Thus, the difference $(m - 1)t_u + c/2$ between $t_{p\text{poss}}$ and t_{poss} is small compared with t_{poss} itself, and PPOSS can almost achieve linear speedup in the running time.

To alleviate the extra $(m - 1)t_u$ running time, i.e., $2b(m - 1)$ comparisons, in the process of updating the population P , we present an accelerating strategy. Lines 10-13 of PPOSS is put into the parallelization process, and “ $P_i = (P \setminus Q) \cup \{s'_i\}$ ” is changed to “ $P_i = P \setminus Q$ ” for each processor. Let $R = \{s'_i \mid \nexists z \in P : z \succ s'_i\}$ record the newly generated solutions satisfying the condition of line 10. After parallelization, we compute the largest non-dominated subset of R containing the new solutions which should be kept; this requires at most $|R|(|R| - 1)/2$ comparisons. By combining this set and $\cap_{i=1}^m P_i$, i.e., the previously archived solutions which should be kept, we get the next population P . Thus, this strategy decreases the number of comparisons from $2b(m - 1)$ to $|R|(|R| - 1)/2$. Note that $|R|$ is usually much smaller than m , because the solutions in P become better as the algorithm runs and the newly generated solutions are easily dominated. In the experiments, this accelerating strategy is used.

The extra running time of each iteration for PPOSS is actually due to the cost of synchronization. The term $c/2$ is because we need to wait until all processors finish, and the extra number $2b(m - 1)$ or $|R|(|R| - 1)/2$ of comparisons is because we need to make P maintain non-dominated solutions generated-so-far. They can be avoided by running the lock-free version of PPOSS, called PPOSS-lf, which performs lines 4-10 of Algorithm 14.2 at each processor asynchronously and independently, but shares an iteration counter. Note that the population P is updated without locks. We will test the performance of PPOSS-lf in the experiments, and leave the theoretical analysis for future.

18.3 Empirical Study

In this section, we conduct experiments on the application of sparse regression, as introduced in Section 14.1.5, to examine the performance of PPOSS on 6 data sets in Table 14.1. We set the budget $b = 8$, and for PPOSS, we use the setting suggested by Theorem 18.1: $T = \lfloor 2eb^2n/m \rfloor$. We test the number m of cores from 1 to 10. All of the experiments are coded in Java and run on an identical configuration: a server with 4 Intel(R) Xeon(R) CPU E5-2640 v2 (8 real cores each, 20MB Cache, 2.00GHz, hyper-threading) and 32GB of RAM. The kernel is Linux 2.6.18-194.el5.

We compare the speed up as well as the solution quality measured by R^2 values with the different number m of cores. The solution quality is also

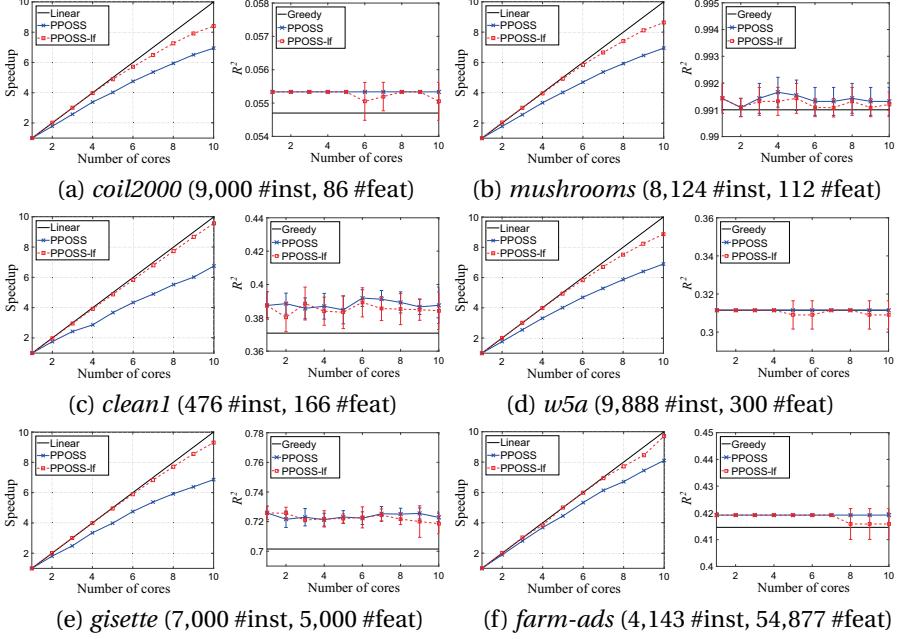


Figure 18.2. On each data set, left: speedup, right: the average and standard deviation of the R^2 value (the larger, the better).

compared with that of the greedy algorithm, which is the existing algorithm with the best-known guarantee [Das and Kempe, 2011].

For PPOSS with each value of m on each data set, we repeat 10 runs independently and report the average speed up and R^2 values. The results are plotted in Figure 18.2. From the left plots in the subfigure of each data set, it can be observed that PPOSS achieves speedup around 7 when the number of cores is 10. From the right plots in each subfigure, it can be seen that the R^2 values of PPOSS with a different number of cores are stable, better than that of the greedy algorithm except that they are the same on the data set *w5a*. Note that on some data sets, e.g., *coil2000*, the standard deviation of the R^2 value by PPOSS is 0, because PPOSS always converges to the same good solutions in 10 runs.

We also test the performance of PPOSS-If, as in Figure 18.2. In the running of PPOSS-If, all cores share an iteration counter, and terminate until the counter reaches $\lceil 2eb^2n \rceil$, i.e., the number of iterations of POSS. As expected, PPOSS-If achieves better speedup than PPOSS, because the extra cost of each iteration in the synchronous setting is avoided. Meanwhile, the R^2 values obtained by PPOSS-If are slightly worse than that of PPOSS because of miss-synchronization.

18.4 Summary

In this chapter, we present the parallel Pareto optimization algorithm PPOSS for subset selection. Theoretically, we prove that for subset selection with monotone objective functions, PPOSS has excellent properties for parallelization while preserving the approximation quality: When the number of processors is limited, almost linear speedups can be achieved; with the increasing number of processors, the running time can be further reduced, eventually to a constant. Note that parallel greedy methods could not enjoy many processors, because it is not easy to make the number of greedy steps parallelizable. This implies that, given sufficient processors, PPOSS can be both faster and more accurate than parallel greedy methods. Empirical results verify our theoretical analysis, and also show that lock-free PPOSS can further improve the speedup with little loss of performance.

A

Appendix

A.1 Proofs in Chapter 4

Proof of Lemma 4.1.

Let π^k denote the distribution of the chain ξ^k . Considering the chain ξ^k at time $k - 1$, because it will be mapped into the space \mathcal{Y} at time k via ϕ , by Lemma 2.2, we have

$$\begin{aligned} \mathbb{E}[\tau^k | \xi_{k-1}^k \sim \pi_{k-1}^k] &= 1 - \pi_{k-1}^k(\mathcal{X}^*) \\ &+ \sum_{x \in \mathcal{X} \setminus \mathcal{X}^*, y \in \mathcal{Y}} \pi_{k-1}^k(x) P(\xi_k^k = y | \xi_{k-1}^k = x) \mathbb{E}[\tau^k | \xi_k^k = y]. \end{aligned} \quad (\text{A.1})$$

The chain ξ^k at time $(k - 1)$ acts like the chain ξ , implying $P(\xi_k^k = y | \xi_{k-1}^k = x) = P(\xi_k \in \phi^{-1}(y) | \xi_{k-1} = x)$. It acts like the chain ξ' from time k , implying $\mathbb{E}[\tau^k | \xi_k^k = y] = \mathbb{E}[\tau' | \xi'_0 = y]$. Thus, we have

$$\begin{aligned} \mathbb{E}[\tau^k | \xi_{k-1}^k \sim \pi_{k-1}^k] &= 1 - \pi_{k-1}^k(\mathcal{X}^*) \\ &+ \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_{k-1}^k(x) P(\xi_k \in \phi^{-1}(y) | \xi_{k-1} = x) \mathbb{E}[\tau' | \xi'_0 = y] \\ &- \sum_{x \in \mathcal{X}^*, y \in \mathcal{Y}} \pi_{k-1}^k(x) P(\xi_k \in \phi^{-1}(y) | \xi_{k-1} = x) \mathbb{E}[\tau' | \xi'_0 = y]. \end{aligned} \quad (\text{A.2})$$

Note that the last minus term of Eq. (A.2) is necessary. This is because if $\xi_{k-1}^k \in \mathcal{X}^*$, the chain should stop running, but the right-aligned mapping may map states in \mathcal{X}^* to $\mathcal{Y} \setminus \mathcal{Y}^*$ and make the chain ξ_{k-1}^k continue to run, which is thus excluded by the last minus term of Eq. (A.2).

By Lemma 2.2, we have

$$\begin{aligned} \mathbb{E}[\tau^k | \xi_0^k \sim \pi_0^k] &= 1 - \pi_0^k(\mathcal{X}^*) + \mathbb{E}[\tau^k | \xi_1^k \sim \pi_1^k] \\ &= \dots = (k - 1) - \sum_{t=0}^{k-2} \pi_t^k(\mathcal{X}^*) + \mathbb{E}[\tau^k | \xi_{k-1}^k \sim \pi_{k-1}^k]. \end{aligned} \quad (\text{A.3})$$

Applying Eq. (A.2) to Eq. (A.3), we have

$$\begin{aligned} \mathbb{E}[\tau^k \mid \xi_0^k \sim \pi_0^k] &= k - \sum_{t=0}^{k-1} \pi_t^k(\mathcal{X}^*) \\ &+ \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_{k-1}^k(x) P(\xi_k \in \phi^{-1}(y) \mid \xi_{k-1} = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ &- \sum_{x \in \mathcal{X}^*, y \in \mathcal{Y}} \pi_{k-1}^k(x) P(\xi_k \in \phi^{-1}(y) \mid \xi_{k-1} = x) \mathbb{E}[\tau' \mid \xi'_0 = y]. \end{aligned} \quad (\text{A.4})$$

Because the chain ξ^k and ξ are identical before time k , we have $\forall t < k : \pi_t^k = \pi_t$, applying which to Eq. (A.4) obtains the lemma. \square

Proof of Theorem 4.1 (“ \geq ” case).

The “ \geq ” case requires a left-aligned mapping. Its proof is similar to that of the “ \leq ” case, and easier because the last minus term of Eq. (A.2) is 0.

As ϕ is a left-aligned mapping, $\mathcal{X}^* \subseteq \phi^{-1}(\mathcal{Y}^*)$, and thus, $\pi_t(\mathcal{X}^*) \leq \pi_t(\phi^{-1}(\mathcal{Y}^*)) = \pi_t^\phi(\mathcal{Y}^*)$. The theorem is again proved by induction. Initialization is the same as that for the “ \leq ” case. Inductive hypothesis assumes that $\forall k \leq K-1 (K \geq 1)$,

$$\mathbb{E}[\tau^k \mid \xi_0^k \sim \pi_0] \geq \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] + \sum_{t=0}^{k-1} \rho_t. \quad (\text{A.5})$$

Applying Lemma 4.1, we have

$$\begin{aligned} \mathbb{E}[\tau^K \mid \xi_0^K \sim \pi_0] &= K - \sum_{t=0}^{K-1} \pi_t(\mathcal{X}^*) \\ &+ \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_{K-1}(x) P(\xi_K \in \phi^{-1}(y) \mid \xi_{K-1} = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ &\geq K - \sum_{t=0}^{K-2} \pi_t(\mathcal{X}^*) - \pi_{K-1}^\phi(\mathcal{Y}^*) + \rho_{K-1} \\ &+ \sum_{u, y \in \mathcal{Y}} \pi_{K-1}^\phi(u) P(\xi'_1 = y \mid \xi'_0 = u) \mathbb{E}[\tau' \mid \xi'_1 = y], \end{aligned} \quad (\text{A.6})$$

where Eq. (A.6) holds by Eq. (4.1) with “ \geq ” and $\pi_{K-1}(\mathcal{X}^*) \leq \pi_{K-1}^\phi(\mathcal{Y}^*)$. Meanwhile, by Lemmas 2.2 and 4.1, we have

$$\begin{aligned} \mathbb{E}[\tau^{K-1} \mid \xi_0^{K-1} \sim \pi_0] &= K - \sum_{t=0}^{K-2} \pi_t(\mathcal{X}^*) - \pi_{K-1}^\phi(\mathcal{Y}^*) \\ &+ \sum_{u, y \in \mathcal{Y}} \pi_{K-1}^\phi(u) P(\xi'_1 = y \mid \xi'_0 = u) \mathbb{E}[\tau' \mid \xi'_1 = y]. \end{aligned} \quad (\text{A.7})$$

Thus, we have

$$\begin{aligned}\mathbb{E}[\tau^K \mid \xi_0^K \sim \pi_0] &\geq \mathbb{E}[\tau^{K-1} \mid \xi_0^{K-1} \sim \pi_0] + \rho_{K-1} \\ &\geq \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] + \sum_{t=0}^{K-1} \rho_t,\end{aligned}\quad (\text{A.8})$$

which proves the induction. Therefore, we come to the conclusion that the theorem holds, using the argument of the “ \leq ” case. \square

Proof of Lemma 4.2.

We prove $\forall 0 \leq j < n : \mathbb{E}(j) < \mathbb{E}(j+1)$ inductively on j .

- (a) **Initialization** is to prove $\mathbb{E}(0) < \mathbb{E}(1)$. Because $\mathbb{E}(1) = 1 + p(1 - p)^{n-1}\mathbb{E}(0) + (1 - p(1-p)^{n-1})\mathbb{E}(1)$, we have $\mathbb{E}(1) = 1/(p(1-p)^{n-1}) > 0 = \mathbb{E}(0)$.
- (b) **Inductive Hypothesis** assumes that

$$\forall 0 \leq j < K (K \leq n-1) : \mathbb{E}(j) < \mathbb{E}(j+1). \quad (\text{A.9})$$

Consider $j = K$. Let x and x' be a solution with $(K+1)$ number of 0-bits and that with K number of 0-bits, respectively. Thus, $\mathbb{E}(K+1) = \mathbb{E}[\tau' \mid \xi'_t = x]$ and $\mathbb{E}(K) = \mathbb{E}[\tau' \mid \xi'_t = x']$. For a Boolean string of length $(n-1)$ with K number of 0-bits, let P_i ($0 \leq i \leq n-1$) denote the probability for the number of 0-bits to become i after bit-wise mutation on this string.

For the solution x , the influence of mutation is divided into two parts: mutation on one 0-bit and mutation on the remaining $(n-1)$ bits. The remaining $(n-1)$ bits contain K number of 0-bits as $|x|_0 = K+1$. Considering the mutation and selection behavior of (1+1)-EA on OneMax, we have

$$\begin{aligned}\mathbb{E}(K+1) &= 1 + p \cdot \left(\sum_{i=0}^{K+1} P_i \mathbb{E}(i) + \sum_{i=K+2}^{n-1} P_i \mathbb{E}(K+1) \right) \\ &\quad + (1-p) \cdot \left(\sum_{i=0}^K P_i \mathbb{E}(i+1) + \sum_{i=K+1}^{n-1} P_i \mathbb{E}(K+1) \right),\end{aligned}\quad (\text{A.10})$$

where p is the probability of flipping the 0-bit in the first mutation part.

For the solution x' , the influence of mutation is also divided into two parts: mutation on one 1-bit and mutation on the remaining $(n-1)$ bits. The remaining $(n-1)$ bits contain K number of 0-bits as $|x'|_0 = K$. Thus,

$$\begin{aligned}\mathbb{E}(K) &= 1 + p \cdot \left(\sum_{i=0}^{K-1} P_i \mathbb{E}(i+1) + \sum_{i=K}^{n-1} P_i \mathbb{E}(K) \right) \\ &\quad + (1-p) \cdot \left(\sum_{i=0}^K P_i \mathbb{E}(i) + \sum_{i=K+1}^{n-1} P_i \mathbb{E}(K) \right),\end{aligned}\quad (\text{A.11})$$

where p is the probability of flipping the 1-bit in the first mutation part.

By Eqs. (A.10) and (A.11), we have

$$\begin{aligned}
& \mathbb{E}(K+1) - \mathbb{E}(K) \\
&= p \cdot \left(\sum_{i=0}^{K-1} P_i(\mathbb{E}(i) - \mathbb{E}(i+1)) + \sum_{i=K+1}^{n-1} P_i(\mathbb{E}(K+1) - \mathbb{E}(K)) \right) \\
&\quad + (1-p) \cdot \left(\sum_{i=0}^K P_i(\mathbb{E}(i+1) - \mathbb{E}(i)) + \sum_{i=K+1}^{n-1} P_i(\mathbb{E}(K+1) - \mathbb{E}(K)) \right) \\
&= (1-2p) \cdot \left(\sum_{i=0}^{K-1} P_i(\mathbb{E}(i+1) - \mathbb{E}(i)) \right) \\
&\quad + \left((1-p)P_K + \sum_{i=K+1}^{n-1} P_i \right) \cdot (\mathbb{E}(K+1) - \mathbb{E}(K)) \\
&> \left((1-p)P_K + \sum_{i=K+1}^{n-1} P_i \right) \cdot (\mathbb{E}(K+1) - \mathbb{E}(K)),
\end{aligned} \tag{A.12}$$

where Eq. (A.12) holds by $0 < p < 0.5$ and inductive hypothesis.

Because $(1-p)P_K + \sum_{i=K+1}^{n-1} P_i < 1$, we have $\mathbb{E}(K+1) > \mathbb{E}(K)$.

(c) Concluding from (a) and (b), the lemma holds. \square

Proof of Lemma 4.4.

Let $f(x_0, \dots, x_m) = \sum_{i=0}^m E_i x_i$. Due to condition (1), i.e., Eq. (4.20), of E_i increasing with i , f is Schur-concave by Theorem A.3 in Chapter 3 of [Marshall et al., 2011]. Conditions (2) and (3), i.e., Eqs. (4.21) and (4.22), imply that the vector (Q_0, \dots, Q_m) majorizes (P_0, \dots, P_m) . Thus, $f(P_0, \dots, P_m) \geq f(Q_0, \dots, Q_m)$, implying that the lemma holds. \square

A.2 Proofs in Chapter 5

Proof of Lemma 5.3.

For the Peak problem, the $<_f$ -partition can contain only two sets $\{\mathcal{S}_1, \mathcal{S}_2\}$ where $\mathcal{S}_1 = \{0, 1\}^n \setminus \{1^n\}$ and $\mathcal{S}_2 = \{1^n\}$. Therefore, we have $m = 2$ and $\gamma_{1,2} = 1$, leading to $\mathfrak{A}_{FL}^u = \pi_0(\mathcal{S}_1) \cdot (1/v_1) = (1 - 1/2^n) \cdot (1/v_1)$ and $\mathfrak{A}_{FL}^l = (1 - 1/2^n) \cdot (1/u_1)$, where v_1 and u_1 are lower and upper bounds of the transition probability from \mathcal{S}_1 to \mathcal{S}_2 , respectively.

For a solution s with $|s|_1 = n - j$ where $j > 0$, the probability of mutating it to 1^n in one step is $(1/n^j)(1 - 1/n)^{n-j}$, which decreases with j . By the definition, $v_1 \leq \min_{s \in \mathcal{S}_1} P(\xi_{t+1} \in \mathcal{S}_2 \mid \xi_t = s) = 1/n^n$, and $u_1 \geq \max_{s \in \mathcal{S}_1} P(\xi_{t+1} \in \mathcal{S}_2 \mid \xi_t = s) = (1/n)(1 - 1/n)^{n-1}$. Therefore, $\mathfrak{A}_{FL}^u \geq (1 - 1/2^n)n^n$ and $\mathfrak{A}_{FL}^l \leq (1 - 1/2^n)n(n/(n-1))^{n-1}$. \square

Proof of Lemma 5.4.

We choose the reference process to be the OneJump chain with dimension n and parameters $p_i = (1/n)^{n-i}(1 - 1/n)^i \forall i < n$. We use the direct mapping function $\phi(x) = x$, which is optimal-aligned.

We examine Eq. (4.1). $\forall x \notin \mathcal{X}^*$ with $|x|_1 = n - j$ where $j > 0$, we have

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] \\ &= (1 - p_{n-j}) \mathbb{E}_{oj}(j) = \left(1 - \frac{1}{n^j} \left(1 - \frac{1}{n}\right)^{n-j}\right) \mathbb{E}_{oj}(j) \\ &= \sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y]. \end{aligned} \quad (\text{A.13})$$

Therefore, $\forall t : \rho_t^u = \rho_t^l = 0$ is a proper assignment of ρ_t^u and ρ_t^l in Characterization 5.1, and thus $\rho^u = \rho^l = 0$. By switch analysis, we have $\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \leq (\text{and } \geq) \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi]$. Moreover,

$$\begin{aligned} \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] &= \sum_{j=1}^n \frac{\binom{n}{j}}{2^n} \cdot \mathbb{E}_{oj}(j) = \sum_{j=1}^n \frac{\binom{n}{j}}{2^n} \cdot n^j \left(\frac{n}{n-1}\right)^{n-j} \\ &= \frac{1}{2^n} \left(\left(n + \frac{n}{n-1}\right)^n - \left(\frac{n}{n-1}\right)^n \right). \end{aligned} \quad (\text{A.14})$$

Thus, we have

$$\begin{aligned} \mathfrak{A}_{SA}^u &= \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] = \frac{1}{2^n} \left(\left(n + \frac{n}{n-1}\right)^n - \left(\frac{n}{n-1}\right)^n \right) \\ &\leq \left(\frac{n}{2} + \frac{n}{2(n-1)}\right)^n, \end{aligned} \quad (\text{A.15})$$

$$\begin{aligned} \mathfrak{A}_{SA}^l &= \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] = \frac{1}{2^n} \left(\left(n + \frac{n}{n-1}\right)^n - \left(\frac{n}{n-1}\right)^n \right) \\ &\geq \left(\frac{n}{2}\right)^n, \end{aligned} \quad (\text{A.16})$$

which prove the lemma. \square

Proof of Lemma 5.8.

Let the initial solution have j ($1 \leq j \leq n$) 0-bits, which are placed randomly. During the running of RLS \neq , the number of 0-bits of the maintained solution never increases and the 0-bits do not change places, because RLS \neq flips one bit at a time, and turning an 1-bit to 0 does not increase the fitness and will be rejected by the strict selection strategy. For one step of RLS \neq , with probability $1/n$, a solution with $(j-1)$ 0-bits will be generated to replace the maintained solution, i.e., the first 0-bit is flipped; with probability $1 - 1/n$, the maintained solution keeps unchanged. Thus, the expected

number of steps to decrease the number of 0-bits by 1 is n . Through step-wise improvements, the expected running time to get to the optimal solution from a solution with j 0-bits is $n \cdot j$. Note that “ $\forall t \geq 0$ ” holds because the process is homogeneous, i.e., the process would be the same if starting from another time point. \square

Proof of Theorem 5.5.

We construct the reference process ξ' by running RLS $^\neq$ on the LeadIn Ones problem of size rn . Thus, $\mathcal{Y} = \{0, 1\}^{rn}$ and $\mathcal{Y}^* = \{1^{rn}\}$. It is then required to construct a mapping function from $\mathcal{X} = \{0, 1, \dots, r\}^n$ to $\mathcal{Y} = \{0, 1\}^{rn}$. Given a discrete linear problem with weights $w_1 \leq \dots \leq w_n$, $\forall \mathbf{s} \in \{0, 1, \dots, r\}^n$, let $\delta(\mathbf{s}, j) = \sum_{i=1}^n w_i s_i - r \sum_{i=j}^n w_i$, and $\theta(\mathbf{s}) = \min\{j \in [n+1] \mid \delta(\mathbf{s}, j) \geq 0\}$, i.e., the threshold index for the sum of the last weights to be no larger than the fitness value. Note that for $\sum_{i=j}^n w_i$, the sum from $j = n+1$ to n is 0. Here, a state $\mathbf{x} \in \mathcal{X}$ is just a solution in the space $\{0, 1, \dots, r\}^n$. For two solutions \mathbf{x} and \mathbf{x}' with $f(\mathbf{x}) \leq f(\mathbf{x}')$, we have $\theta(\mathbf{x}) \geq \theta(\mathbf{x}')$. Let

$$m(\mathbf{x}) = r(n - \theta(\mathbf{x}) + 1) + \lfloor \delta(\mathbf{x}, \theta(\mathbf{x}))/w_{\theta(\mathbf{x})-1} \rfloor. \quad (\text{A.17})$$

For \mathbf{x} and \mathbf{x}' with $f(\mathbf{x}) \leq f(\mathbf{x}')$, we have $m(\mathbf{x}) \leq m(\mathbf{x}')$. This is because, when $\theta(\mathbf{x}) = \theta(\mathbf{x}')$, denoting $a = w_{\theta(\mathbf{x})-1}$ and $b = r \sum_{i=\theta(\mathbf{x})}^n w_i$, we have

$$\begin{aligned} m(\mathbf{x}) - m(\mathbf{x}') &= \lfloor \delta(\mathbf{x}, \theta(\mathbf{x}))/a \rfloor - \lfloor \delta(\mathbf{x}', \theta(\mathbf{x}))/a \rfloor \\ &= \lfloor (f(\mathbf{x}) - b)/a \rfloor - \lfloor (f(\mathbf{x}') - b)/a \rfloor \leq 0; \end{aligned} \quad (\text{A.18})$$

when $\theta(\mathbf{x}) \geq \theta(\mathbf{x}') + 1$, noting that $\delta(\mathbf{x}, \theta(\mathbf{x})) < rw_{\theta(\mathbf{x})-1}$ because otherwise $\theta(\mathbf{x})$ is not the minimum index, we have

$$m(\mathbf{x}) - m(\mathbf{x}') \leq -r + \left\lfloor \frac{\delta(\mathbf{x}, \theta(\mathbf{x}))}{w_{\theta(\mathbf{x})-1}} \right\rfloor - \left\lfloor \frac{\delta(\mathbf{x}', \theta(\mathbf{x}'))}{w_{\theta(\mathbf{x}')-1}} \right\rfloor \leq -r + r = 0. \quad (\text{A.19})$$

The mapping function is $\forall \mathbf{x} : \phi(\mathbf{x}) = 1^{m(\mathbf{x})} 0^{rn-m(\mathbf{x})}$. The target space $\mathcal{X}^* = \{r^n\}$. This function is optimal-aligned, because $\theta(r^n) = 1$, $m(r^n) = rn$ and thus $\phi(r^n) = 1^{rn}$, while vice versa.

We examine Eq. (4.1). $\forall \mathbf{x} \notin \mathcal{X}^*$, we have

$$\begin{aligned} \sum_{y \in \mathcal{Y}} P(\xi'_1 = y \mid \xi'_0 = \phi(\mathbf{x})) \mathbb{E}[\tau' \mid \xi'_1 = y] \\ = \mathbb{E}_{rls}(rn - m(\mathbf{x})) - 1 = rn(rn - m(\mathbf{x})) - 1. \end{aligned} \quad (\text{A.20})$$

For the process ξ , let \mathbf{x}' be the next solution after mutating \mathbf{x} and passing the selection step. By the behavior of selection, we have $f(\mathbf{x}') \geq f(\mathbf{x})$, implying $m(\mathbf{x}') \geq m(\mathbf{x})$. As \mathbf{x} is non-optimal, there is at least one $j \in \{\theta(\mathbf{x})-1, \theta(\mathbf{x}), \dots, n\}$ such that the j -th element of \mathbf{x} is smaller than r . Thus, the probability of $m(\mathbf{x}') \geq m(\mathbf{x}) + 1$ is at least $(1/r)p(1-p)^{n-1}$, because it

is sufficient to flip the j -th element of \mathbf{x} to r and keep other elements unchanged. Because $\mathbb{E}_{rls}(i)$ increases with i , we have

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = \mathbf{x}) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ & \leq \frac{1}{r} p(1-p)^{n-1} \cdot \mathbb{E}_{rls}(rn - m(\mathbf{x}) - 1) + \left(1 - \frac{1}{r} p(1-p)^{n-1}\right) \cdot \mathbb{E}_{rls}(rn - m(\mathbf{x})) \\ & = rn(rn - m(\mathbf{x})) - np(1-p)^{n-1}. \end{aligned} \quad (\text{A.21})$$

Combining Eqs. (A.20) and (A.21), we have

$$\begin{aligned} & \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_t(x) P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ & \leq \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_t(x) P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] \\ & \quad + (1 - np(1-p)^{n-1}) \cdot (1 - \pi_t(\mathcal{X}^*)), \end{aligned} \quad (\text{A.22})$$

implying that $\forall t : \rho_t^u = (1 - np(1-p)^{n-1}) \cdot (1 - \pi_t(\mathcal{X}^*))$ is a proper assignment of ρ_t^u in Characterization 5.1. Considering $\sum_{t=0}^{+\infty} (1 - \pi_t(\mathcal{X}^*)) = \mathbb{E}[\tau \mid \xi_0 \sim \pi_0]$, we have

$$\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \leq \frac{\mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi]}{np(1-p)^{n-1}}. \quad (\text{A.23})$$

For the reference process, we have $\mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] \leq \mathbb{E}_{rls}(rn) = r^2 n^2$. Thus,

$$\mathfrak{A}_{SA}^u = \frac{\mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi]}{np(1-p)^{n-1}} \leq \frac{r^2 n}{p(1-p)^{n-1}}, \quad (\text{A.24})$$

which proves the theorem. \square

Proof of Lemma 5.11.

We first prove $\forall i \geq 1 : \mathbb{E}_{tj}(i) \geq 1/(p^i(1-p)^{n-i})$ inductively on i .

(a) Initialization is to prove $\mathbb{E}_{tj}(n) \geq 1/p^n$. Because $\mathbb{E}_{tj}(n) = 1 + p^n \mathbb{E}_{tj}(0) + (1 - p^n) \mathbb{E}_{tj}(n)$, we have $\mathbb{E}_{tj}(n) = 1/p^n$.

(b) Inductive Hypothesis assumes that

$$\forall i > K \ (K \leq n-1) : \mathbb{E}_{tj}(i) \geq \frac{1}{p^i(1-p)^{n-i}}. \quad (\text{A.25})$$

Consider $i = K$. For $K \geq 2$, by Eq. (5.31) and Lemma 2.1, we have

$$\begin{aligned} & \mathbb{E}_{tj}(K) \\ & = 1 + p^K (1-p)^{n-K} \mathbb{E}_{tj}(0) + (n-K)p(1-p)^{n-1} \mathbb{E}_{tj}(K+1) \\ & \quad + (1 - p^K (1-p)^{n-K} - (n-K)p(1-p)^{n-1}) \mathbb{E}_{tj}(K) \end{aligned}$$

$$= \frac{1 + (n - K)p(1 - p)^{n-1}\mathbb{E}_{tj}(K + 1)}{p^K(1 - p)^{n-K} + (n - K)p(1 - p)^{n-1}} \quad (\text{A.26})$$

$$\geq \frac{p^K(1 - p)^{n-K} + (n - K)p(1 - p)^{n-1}\frac{1-p}{p}}{p^K(1 - p)^{n-K} + (n - K)p(1 - p)^{n-1}} \cdot \frac{1}{p^K(1 - p)^{n-K}} \quad (\text{A.27})$$

$$\geq \frac{1}{p^K(1 - p)^{n-K}}, \quad (\text{A.28})$$

where Eq. (A.27) holds because $\mathbb{E}_{tj}(K + 1) \geq 1/(p^{K+1}(1 - p)^{n-K-1})$ by inductive hypothesis, and Eq. (A.28) holds by $p < 0.5$. For $K = 1$, by Eq. (5.32) and Lemma 2.1, we have

$$\begin{aligned} \mathbb{E}_{tj}(1) &= 1 + p(1 - p)^{n-1}\mathbb{E}_{tj}(0) + p(1 - p)^{n-1}\mathbb{E}_{tj}(2) + (1 - 2p(1 - p)^{n-1})\mathbb{E}_{tj}(1) \\ &= \frac{1 + p(1 - p)^{n-1}\mathbb{E}_{tj}(2)}{2p(1 - p)^{n-1}} \end{aligned} \quad (\text{A.29})$$

$$\begin{aligned} &\geq \frac{p(1 - p)^{n-1} + (1 - p)^n}{2p(1 - p)^{n-1}} \cdot \frac{1}{p(1 - p)^{n-1}} \\ &\geq \frac{1}{p(1 - p)^{n-1}}. \end{aligned} \quad (\text{A.30})$$

(c) Concluding from (a) and (b), it holds that

$$\forall i \geq 1 : \mathbb{E}_{tj}(i) \geq \frac{1}{p^i(1 - p)^{n-i}}. \quad (\text{A.31})$$

Next we show that $\forall i \geq 1 : \mathbb{E}_{tj}(i - 1) \leq \mathbb{E}_{tj}(i)$. It holds that $\mathbb{E}_{tj}(0) = 0 < \mathbb{E}_{tj}(1)$. By Eq. (A.29), we have

$$\begin{aligned} \mathbb{E}_{tj}(1) &= \frac{1 + p(1 - p)^{n-1}\mathbb{E}_{tj}(2)}{2p(1 - p)^{n-1}} \\ &\leq \frac{p^2(1 - p)^{n-2} + p(1 - p)^{n-1}}{2p(1 - p)^{n-1}} \cdot \mathbb{E}_{tj}(2) \\ &\leq \mathbb{E}_{tj}(2). \end{aligned} \quad (\text{A.32})$$

For $i \geq 2$, by Eq. (A.26), we have

$$\begin{aligned} \mathbb{E}_{tj}(i) &= \frac{1 + (n - i)p(1 - p)^{n-1}\mathbb{E}_{tj}(i + 1)}{p^i(1 - p)^{n-i} + (n - i)p(1 - p)^{n-1}} \\ &\leq \frac{p^{i+1}(1 - p)^{n-i-1} + (n - i)p(1 - p)^{n-1}}{p^i(1 - p)^{n-i} + (n - i)p(1 - p)^{n-1}} \cdot \mathbb{E}_{tj}(i + 1) \\ &\leq \mathbb{E}_{tj}(i + 1). \end{aligned} \quad (\text{A.33})$$

□

Proof of Theorem 5.7.

Let the chain $\xi \in \mathcal{X}$ model the concerned process. Thus, $\mathcal{X} = \{0, 1\}^n$ and $\mathcal{X}^* = \{0^{n-1}1\}$. Let $\xi' \in \mathcal{Y}$ model the reference process, i.e., the TrapJump chain with dimension n . Thus, $\mathcal{Y} = \{0, 1, \dots, n\}$ and $\mathcal{Y}^* = \{0\}$. The parameter p of TrapJump is set as the mutation probability of (1+1)-EA.

We divide \mathcal{X} into $\{\mathcal{X}^*, \mathcal{X}_0^F, \dots, \mathcal{X}_{n-1}^F, \mathcal{X}^I\}$, where \mathcal{X}_i^F contains feasible solutions having Hamming distance $(n-i)$ from the optimal solution $0^{n-1}1$, and \mathcal{X}^I contains all infeasible solutions. In other words, $\mathcal{X}_i^F = \{(s, 0) \mid s \in \{0, 1\}^{n-1}, |s|_1 = n-1-i\}$ and $\mathcal{X}^I = \{(s, 1) \mid s \in \{0, 1\}^{n-1}, |s|_1 > 0\}$. We construct the mapping function $\phi : \mathcal{X} \rightarrow \mathcal{Y}$ as

$$\phi(x) = \begin{cases} 0 & \text{if } x \in \mathcal{X}^*; \\ n-i & \text{if } x \in \mathcal{X}_i^F; \\ 1 & \text{if } x \in \mathcal{X}^I. \end{cases} \quad (\text{A.34})$$

ϕ is an optimal-aligned mapping because $\phi(x) \in \mathcal{Y}^* = \{0\}$ iff $x \in \mathcal{X}^*$.

We examine the condition, i.e., Eq. (4.1), of switch analysis. We first calculate the left part of Eq. (4.1). For ξ , any solution $x \in \mathcal{X}_i^F$ can jump only to $\mathcal{X}^* \cup \mathcal{X}_0^F \cup \dots \cup \mathcal{X}_{i-1}^F$, because (1+1)-EA accepts only better offspring solutions and

$$f(x \in \mathcal{X}^*) > f(x \in \mathcal{X}_0^F) > \dots > f(x \in \mathcal{X}_{n-1}^F) > f(x \in \mathcal{X}^I). \quad (\text{A.35})$$

$\forall x \in \mathcal{X}_i^F$, $P(\xi_{t+1} \in \mathcal{X}^* \mid \xi_t = x) = p^{n-i}(1-p)^i$ because it has Hamming distance $(n-i)$ from the optimal solution; $P(\xi_{t+1} \in \mathcal{X}_{i-1}^F \mid \xi_t = x) \geq (1-p) \cdot ip(1-p)^{n-2}$ because it is sufficient to keep the last 0-bit unchanged, i.e., keep the solution feasible, flip one of the other i 0-bits and keep the remaining bits unchanged. Considering $\mathbb{E}_{tj}(i)$ increasing with i by Lemma 5.11, we have, $\forall x \in \mathcal{X}_i^F$,

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ & \geq p^{n-i}(1-p)^i \mathbb{E}_{tj}(0) + ip(1-p)^{n-1} \mathbb{E}_{tj}(n-i+1) \\ & \quad + (1-p^{n-i}(1-p)^i - ip(1-p)^{n-1}) \mathbb{E}_{tj}(n-i). \end{aligned} \quad (\text{A.36})$$

$\forall x \in \mathcal{X}^I$, $P(\xi_{t+1} \in \mathcal{X}^* \mid \xi_t = x) \leq p(1-p)^{n-1}$ because the Hamming distance from the optimal solution is at least 1; $P(\xi_{t+1} \in \mathcal{X}_0^F \cup \dots \cup \mathcal{X}_{n-2}^F \mid \xi_t = x) \geq p(1-p)^{n-1}$ because it is sufficient to flip the last 1-bit and keep the other bits unchanged. Thus, we have, $\forall x \in \mathcal{X}^I$,

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ & \geq p(1-p)^{n-1} \mathbb{E}_{tj}(0) + p(1-p)^{n-1} \mathbb{E}_{tj}(2) \\ & \quad + (1-2p(1-p)^{n-1}) \mathbb{E}_{tj}(1). \end{aligned} \quad (\text{A.37})$$

We then calculate the right part of Eq. (4.1). $\forall x \in \mathcal{X}_i^F$ with $i < n - 1$, by $\phi(x) = n - i > 1$ and Eq. (5.31), we have

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] \\ &= p^{n-i}(1-p)^i \mathbb{E}_{tj}(0) + ip(1-p)^{n-1} \mathbb{E}_{tj}(n-i+1) \\ &+ (1-p^{n-i}(1-p)^i - ip(1-p)^{n-1}) \mathbb{E}_{tj}(n-i). \end{aligned} \quad (\text{A.38})$$

$\forall x \in \mathcal{X}_{n-1}^F \cup \mathcal{X}^I$, by $\phi(x) = 1$ and Eq. (5.32), we have

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] \\ &= p(1-p)^{n-1} \mathbb{E}_{tj}(0) + p(1-p)^{n-1} \mathbb{E}_{tj}(2) \\ &+ (1-2p(1-p)^{n-1}) \mathbb{E}_{tj}(1). \end{aligned} \quad (\text{A.39})$$

Combining Eqs. (A.36) to (A.39), we have

$$\begin{aligned} & \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_t(x) P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\ & \geq \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_t(x) P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y], \end{aligned} \quad (\text{A.40})$$

implying $\rho^l = 0$ in Characterization 5.1. Thus, we have

$$\mathfrak{A}_{SA}^l = \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi]. \quad (\text{A.41})$$

Next, we examine $\mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi]$. Because the initial distribution is uniform over $\mathcal{X} = \{0, 1\}^n$, $\pi_0(\mathcal{X}^*) = 1/2^n$, $\pi_0(\mathcal{X}_i^F) = \binom{n-1}{n-1-i}/2^n$ and $\pi_0(\mathcal{X}^I) = 1/2 - 1/2^n$. Thus, we have

$$\begin{aligned} \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] &= \sum_{i=0}^n \pi_0^\phi(i) \mathbb{E}_{tj}(i) \\ &= \pi_0(\mathcal{X}_{n-1}^F \cup \mathcal{X}^I) \mathbb{E}_{tj}(1) + \sum_{i=2}^n \pi_0(\mathcal{X}_{n-i}^F) \mathbb{E}_{tj}(i) \\ &\geq \frac{1}{2p(1-p)^{n-1}} + \sum_{i=2}^n \frac{\binom{n-1}{i-1}}{2^n} \frac{1}{p^i(1-p)^{n-i}} \end{aligned} \quad (\text{A.42})$$

$$\begin{aligned} &= \frac{1}{2^n p^n (1-p)^{n-1}} + \frac{1}{2p(1-p)^{n-1}} - \frac{1}{2^n p(1-p)^{n-1}} \\ &= \Omega\left(\left(\frac{1}{2p(1-p)}\right)^n\right), \end{aligned} \quad (\text{A.43})$$

where Eq. (A.42) holds by Lemma 5.11. Combining Eqs. (A.41) and (A.43), the theorem holds. \square

Proof of Theorem 5.8.

Let the chain $\xi \in \mathcal{X}$ model the concerned process of RLS running on the constrained Trap problem. The reference process ξ' and the mapping function ϕ are the same as those used in the proof of Theorem 5.7, except that the parameter p of TrapJump is set as $1/n$.

We examine the condition, i.e., Eq. (4.1), of switch analysis. We first calculate the left part of Eq. (4.1). $\forall x \in \mathcal{X}_{n-1}^F$, $P(\xi_{t+1} \in \mathcal{X}^* | \xi_t = x) = 1/n$ and $P(\xi_{t+1} \in \mathcal{X}_{n-2}^F | \xi_t = x) = 1 - 1/n$, because the offspring solution generated by one-bit mutation belongs to either \mathcal{X}^* or \mathcal{X}_{n-2}^F , and will always be accepted. Thus, we have, $\forall x \in \mathcal{X}_{n-1}^F$,

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) | \xi_t = x) \mathbb{E}[\tau' | \xi'_0 = y] \\ &= \frac{1}{n} \mathbb{E}_{tj}(0) + \left(1 - \frac{1}{n}\right) \mathbb{E}_{tj}(2). \end{aligned} \quad (\text{A.44})$$

$\forall x \in \mathcal{X}_i^F$ with $i < n - 1$, we have $P(\xi_{t+1} \in \mathcal{X}_{i-1}^F | \xi_t = x) = i/n$ and $P(\xi_{t+1} \in \mathcal{X}_i^F | \xi_t = x) = 1 - i/n$, leading to

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) | \xi_t = x) \mathbb{E}[\tau' | \xi'_0 = y] \\ &= \frac{i}{n} \mathbb{E}_{tj}(n-i+1) + \left(1 - \frac{i}{n}\right) \mathbb{E}_{tj}(n-i). \end{aligned} \quad (\text{A.45})$$

$\forall x \in \mathcal{X}_I$, $P(\xi_{t+1} \in \mathcal{X}_F | \xi_t = x) = 1/n$ because $\xi_{t+1} \in \mathcal{X}_F$ iff the last 1-bit of x is flipped. Meanwhile, $P(\xi_{t+1} \in \mathcal{X}^* | \xi_t = x) \leq 1/n$. Thus, $\forall x \in \mathcal{X}_I$,

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) | \xi_t = x) \mathbb{E}[\tau' | \xi'_0 = y] \\ & \geq \frac{1}{n} \mathbb{E}_{tj}(0) + \frac{1}{n} \mathbb{E}_{tj}(2) + \left(1 - \frac{2}{n}\right) \mathbb{E}_{tj}(1). \end{aligned} \quad (\text{A.46})$$

For the right part of Eq. (4.1), Eqs. (A.38) and (A.39) hold with $p = 1/n$. By comparing Eqs. (A.38) and (A.45), we have, $\forall x \in \mathcal{X}_i^F$ with $i < n - 1$,

$$\begin{aligned} & \sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) | \xi_t = x) \mathbb{E}[\tau' | \xi'_0 = y] \\ & \geq \sum_{y \in \mathcal{Y}} P(\xi'_1 = y | \xi'_0 = \phi(x)) \mathbb{E}[\tau' | \xi'_1 = y], \end{aligned} \quad (\text{A.47})$$

where Eq. (A.47) holds by $i/n > ip(1-p)^{n-1} = (i/n)(1-1/n)^{n-1}$ and $\mathbb{E}_{tj}(i)$ increasing with i . By comparing Eqs. (A.39), (A.44) and (A.46), we have, $\forall x \in \mathcal{X}_{n-1}^F \cup \mathcal{X}_I$,

$$\sum_{y \in \mathcal{Y}} P(\xi_{t+1} \in \phi^{-1}(y) | \xi_t = x) \mathbb{E}[\tau' | \xi'_0 = y]$$

$$\begin{aligned}
& - \sum_{y \in \mathcal{Y}} P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] \\
& \geq \frac{1}{n} \mathbb{E}_{tj}(2) + \left(1 - \frac{2}{n}\right) \mathbb{E}_{tj}(1) \\
& \quad - \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} \mathbb{E}_{tj}(2) - \left(1 - \frac{2}{n} \left(1 - \frac{1}{n}\right)^{n-1}\right) \mathbb{E}_{tj}(1) \\
& = \frac{1}{n} \mathbb{E}_{tj}(2) - \frac{2}{n} \mathbb{E}_{tj}(1) + 1 = -\frac{1}{n^{\frac{1}{n}} \left(1 - \frac{1}{n}\right)^{n-1}} + 1 \tag{A.48} \\
& > -2. \tag{A.49}
\end{aligned}$$

Eq. (A.48) holds by Eq. (A.29). Combining Eqs. (A.47) and (A.49) leads to

$$\begin{aligned}
& \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_t(x) P(\xi_{t+1} \in \phi^{-1}(y) \mid \xi_t = x) \mathbb{E}[\tau' \mid \xi'_0 = y] \\
& \quad - \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \pi_t(x) P(\xi'_1 = y \mid \xi'_0 = \phi(x)) \mathbb{E}[\tau' \mid \xi'_1 = y] \\
& \geq -2\pi_t(\mathcal{X}_{n-1}^F \cup \mathcal{X}_I) \\
& \geq -2 \left(1 - \frac{1}{n}\right)^t \pi_0(\mathcal{X}_{n-1}^F \cup \mathcal{X}_I) \tag{A.50}
\end{aligned}$$

$$= - \left(1 - \frac{1}{n}\right)^t, \tag{A.51}$$

where Eq. (A.50) holds by Eqs. (3.39) and (3.45), and Eq. (A.51) holds by $\pi_0(\mathcal{X}_{n-1}^F) = 1/2^n$ and $\pi_0(\mathcal{X}_I) = (2^n - 1)/2^n$.

Eq. (A.51) implies $\rho^l = -\sum_{t=0}^{+\infty} (1 - \frac{1}{n})^t = -n$ in Characterization 5.1. Thus, we have $\mathfrak{A}_{SA}^l = \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0^\phi] - n$. Using Eq. (A.43) with $p = 1/n$, we have $\mathfrak{A}_{SA}^l = \Omega((n/2)^n) - n = \Omega((n/2)^n)$, i.e., the theorem holds. \square

A.3 Proofs in Chapter 8

Proof of Lemma 8.4.

By the definition of COCZ, the objective vector can be represented as $(i + j, i + n/2 - j)$, where i and j ($0 \leq i, j \leq n/2$) are the number of 1-bits in the first and second half of a solution, respectively. For each value of j , there can be only one corresponding value of i , because two objective vectors with the same j value and different i values are comparable, violating the property that the MOEAs maintain non-dominated solutions. Thus, the size of $f(P) = \{f(s) \mid s \in P\}$ is at most $n/2 + 1$.

Because the population P has a one-to-one correspondence with the set $f(P)$ of objective vectors, the population size is no larger than $n/2 + 1$. \square

Proof of Theorem 8.4.

For COCZ, the size of the Pareto front is $n/2 + 1$. After the initialization procedure, the two Pareto optimal solutions 1^n and $1^{\frac{n}{2}}0^{\frac{n}{2}}$ have been found. Thus, it is sufficient to increase the number of Pareto optimal solutions for $(n/2 - 1)$ times to find the Pareto front.

We consider the expected number of steps for a success, i.e., increasing the number of Pareto optimal solutions, when starting from a population P containing $(i + 1)$ Pareto optimal solutions, where $i \in [n/2 - 1]$. In the selection procedure, because the two solutions 1^n and $1^{\frac{n}{2}}0^{\frac{n}{2}}$, which are optimal for the two objectives of COCZ, respectively, have been found, the algorithm actually selects one solution s randomly from $\{1^n, 1^{\frac{n}{2}}0^{\frac{n}{2}}\}$ at first, and then selects the other solution randomly from the remaining solutions $P \setminus \{s\}$. Consider two cases in selection for the probability q of generating one or two new Pareto optimal solutions by one-point recombination in one step: the two solutions 1^n and $1^{\frac{n}{2}}0^{\frac{n}{2}}$ are selected; one selected solution is either 1^n or $1^{\frac{n}{2}}0^{\frac{n}{2}}$ and the other selected one is a Pareto optimal solution from $P \setminus \{1^n, 1^{\frac{n}{2}}0^{\frac{n}{2}}\}$. In the former case occurring with probability $1/(|P| - 1)$, $q \geq (n/2 - i)/(n - 1)$. In the latter case occurring with probability $1/(2(|P| - 1))$, suppose that the Pareto optimal solution selected from $P \setminus \{1^n, 1^{\frac{n}{2}}0^{\frac{n}{2}}\}$ has k ($1 \leq k \leq n/2 - 1$) 0-bits, the number of Pareto optimal solutions having less than k 0-bits in the current population is k' ($1 \leq k' \leq i - 1$) and the other selected solution is 1^n , then $q \geq (k - k')/(n - 1)$. Correspondingly, when the other selected solution is $1^{\frac{n}{2}}0^{\frac{n}{2}}$, $q \geq (n/2 - k - i + k')/(n - 1)$. Thus, in the latter case, $q \geq (k - k')/(n - 1) + (n/2 - k - i + k')/(n - 1) = (n/2 - i)/(n - 1)$. By combining these two cases, we have

$$\begin{aligned} q &\geq \frac{1}{|P| - 1} \cdot \frac{\frac{n}{2} - i}{n - 1} + \frac{i - 1}{2(|P| - 1)} \cdot \frac{\frac{n}{2} - i}{n - 1} \\ &= \frac{(i + 1)(\frac{n}{2} - i)}{2(|P| - 1)(n - 1)}. \end{aligned} \quad (\text{A.52})$$

Because the recombination probability is $1/2$, the number of Pareto optimal solutions increases by one step with probability at least $(i + 1)(n/2 - i)/(4(|P| - 1)(n - 1)) \geq (i + 1)(n/2 - i)/(2n(n - 1))$, where the inequality holds by $|P| \leq n/2 + 1$ from Lemma 8.4. Thus, the expected number of steps for a success is at most $2n(n - 1)/((i + 1)(n/2 - i))$.

Thus, the expected running time to find the Pareto front is at most

$$2 \cdot \sum_{i=1}^{n/2-1} \frac{2n(n - 1)}{(i + 1)(n/2 - i)} = O(n \log n). \quad (\text{A.53})$$

By combining the expected running time $\Theta(n \log n)$ of the initialization process in Lemma 8.3, the expected running time of MOEA_{recomb,0.5}^{onebit} and MOEA_{recomb,0.5}^{bitwise} on COCZ is $\Theta(n \log n)$. \square

Proof of Theorem 8.5.

We divide the evolutionary process into two phases. The first phase starts after initialization and finishes until the first Pareto optimal solution is found. The second phase finishes until the Pareto front is found.

For the first phase, let j ($0 \leq j \leq n/2$) denote the maximum number of 1-bits in the first half of solutions in the current population. Because a solution cannot be dominated by another solution with less 1-bits in the first half, j cannot decrease. j increases by one step with probability at least $(1/(n/2+1)) \cdot ((n/2-j)/n)(1-1/n)^{n-1} \geq (n/2-j)/(en(n/2+1))$, because it is sufficient to select a solution with j 1-bits in its first half for mutation, and flip only one 0-bit in the first half of this solution; the probability of selection is at least $1/(n/2+1)$ by Lemma 8.4 and the probability of mutation is $((n/2-j)/n)(1-1/n)^{n-1}$. Because any solution with $(n/2)$ 1-bits in its first half is a Pareto optimal solution, $n/2$ steps of increasing j are sufficient to find the first Pareto optimal solution. Thus, the expected running time of the first phase is at most $\sum_{j=0}^{n/2-1} en(n/2+1)/(n/2-j) = O(n^2 \log n)$.

For the second phase, before finding the Pareto front, there always exists at least one Pareto optimal solution in the current population, from which one new Pareto optimal solution can be generated by flipping only one 1-bit or one 0-bit in its second half. We call such Pareto optimal solutions *boundary Pareto optimal solutions*. Thus, the number of Pareto optimal solutions can increase by 1 in one step with probability at least $(1/(n/2+1)) \cdot (\min\{i, n/2-i\}/n)(1-1/n)^{n-1} \geq \min\{i, n/2-i\}/(en(n/2+1))$, because it is sufficient to select one boundary Pareto optimal solution for mutation, and flip only one 1-bit or one 0-bit in its second half; the probability of selection is at least $1/(n/2+1)$ by Lemma 8.4 and the probability of mutation is at least $(\min\{i, n/2-i\}/n)(1-1/n)^{n-1}$ where i is the number of 1-bits in the second half of the selected boundary Pareto optimal solution. Because $n/2$ steps of increasing the number of Pareto optimal solutions are sufficient to find the Pareto front of COCZ, the expected running time of the second phase is at most $\sum_{i=1}^{\lceil n/4 \rceil} 2 \cdot (en(n/2+1)/i) = O(n^2 \log n)$.

By combining the running time of these two phases, the expected running time of the whole evolutionary process is $O(n^2 \log n)$. \square

Proof of Theorem 8.6.

For COCZ, we define that two Pareto optimal solutions are *consecutive* if the difference of the number of 0-bits between these two solutions is 1. The offspring Pareto optimal solution generated by one-bit mutation on a Pareto optimal solution must be consecutive with the parent. After initialization, the two Pareto optimal solutions 1^n and $1^{\frac{n}{2}}0^{\frac{n}{2}}$ have been found. Thus, the population in the evolutionary process is always constructed by two continuous sets L and R of Pareto optimal solutions, where $1^n \in L$, $1^{\frac{n}{2}}0^{\frac{n}{2}} \in R$, and every two adjacent Pareto optimal solutions in L or R are consecutive. We divide the optimization process into $n/2$ phases, where the

population in the i -th phase ($1 \leq i \leq n/2$) consists of $(i+1)$ Pareto optimal solutions and the $(n/2)$ -th phase corresponds to the phase where the Pareto front has been found. In the following, we will consider the probability of generating new Pareto optimal solutions by one step in each phase.

In the first phase, $|L| = 1$ and $|R| = 1$. For reproduction, the two solutions 1^n and $1^{\frac{n}{2}}0^{\frac{n}{2}}$ are selected. For 1^n , the offspring solution generated by one-bit mutation will be accepted only if one 1-bit in the second half is mutated. For $1^{\frac{n}{2}}0^{\frac{n}{2}}$, the offspring solution will be accepted only if one 0-bit is mutated. Thus, by one step, two new Pareto optimal solutions will be generated simultaneously with probability $1/4$; only one new Pareto optimal solution will be generated with probability $1/2$; otherwise, no new Pareto optimal solution will be generated.

Next, consider the i -th phase, where $1 < i \leq n/2 - 1$. In the selection procedure, because 1^n and $1^{\frac{n}{2}}0^{\frac{n}{2}}$, which are optimal for the two objectives of COCZ, respectively, have been found, one solution will be randomly selected from $\{1^n, 1^{\frac{n}{2}}0^{\frac{n}{2}}\}$ at first, and the other one will be randomly selected from the remaining solutions. The probabilities for two solutions selected by this procedure are $1/2$ and $1/i$, respectively. There are two cases.

Case 1: $\min\{|L|, |R|\} > 1$. One-bit mutation on either 1^n or $1^{\frac{n}{2}}0^{\frac{n}{2}}$ cannot generate new Pareto optimal solutions, and only that on the rightmost solution in L and the leftmost solution in R can generate one new Pareto optimal solution with probability $(n/2 - (|L| - 1))/n$ and $(n/2 - (|R| - 1))/n$, respectively. Thus, one new Pareto optimal solution can be generated by one step with probability $(n/2 - (|L| - 1))/(in) + (n/2 - (|R| - 1))/(in) = (n-i+1)/(in)$; otherwise, no new Pareto optimal solution will be generated.

Case 2: $\min\{|L|, |R|\} = 1$. Suppose $|L| = 1$. If the solution selected from $\{1^n, 1^{\frac{n}{2}}0^{\frac{n}{2}}\}$ is $1^{\frac{n}{2}}0^{\frac{n}{2}}$, one-bit mutation cannot generate new Pareto optimal solutions, and only when the other selected solution is 1^n or the leftmost solution of R can mutation generate one new Pareto optimal solution with probability $1/2$ and $(n/2 - (i - 1))/n$, respectively. If the solution selected from $\{1^n, 1^{\frac{n}{2}}0^{\frac{n}{2}}\}$ is 1^n , by mutation on 1^n , one new Pareto optimal solution can be generated with probability $1/2$, and only when the other selected solution is the leftmost solution of R can one new Pareto optimal solution be generated by mutation with probability $(n/2 - (i - 1))/n$. Thus, when $i < n/2 - 1$, by one step, only one new Pareto optimal solution will be generated while $\min\{|L|, |R|\}$ remains 1 with probability $3(n/2 - i + 1)/(4in)$; one or two new Pareto optimal solutions will be generated while $\min\{|L|, |R|\} > 1$ with probability $(i + 1)/(4i)$; otherwise, no new Pareto optimal solution will be generated. When $i = n/2 - 1$, the last undiscovered Pareto optimal solution will be found by one step with probability $(n^2 + 12)/(4n^2 - 8n)$.

The state of the population during evolution changes as similar as that in the proof of Theorem 8.2. According to the analysis above, the probabilities of generating one new Pareto optimal solution in the process of steps 3 and 5 are $\Theta((n/2 - i + 1)/(in))$ and $\Theta((n - i + 1)/(in))$, respectively; the probability of transferring at steps 2 and 4 is $\Theta(1)$. Thus, the expected number

of steps after initialization to generate the Pareto front is $\Omega(\sum_{i=1}^{n/2-1} in/(n-i+1)) = \Omega(n^2)$ and $O(\sum_{i=1}^{n/2-1} in/(n/2-i+1)) = O(n^2 \log n)$. By combining the expected running time $\Theta(n \log n)$ of initialization in Lemma 8.3, the total expected running time is $\Omega(n^2)$ and $O(n^2 \log n)$. \square

Proof of Theorem 8.7.

After the initialization procedure, the two Pareto optimal solutions 1^n and $1^{\frac{n}{2}}0^{\frac{n}{2}}$ have been found. Thus, it is sufficient to increase the number of Pareto optimal solutions for $(n/2 - 1)$ times to find the Pareto front.

Before finding the Pareto front, there always exists at least one Pareto optimal solution in the current population, from which one new Pareto optimal solution can be generated by flipping only one 1-bit or one 0-bit in its second half. Because the probability of selecting a specific solution is at least $1/(n/2)$ by Lemma 8.4, the number of Pareto optimal solutions increases by one step with probability at least $(1/(n/2)) \cdot (\min\{i, n/2-i\}/n)(1-1/n)^{n-1} \geq \min\{i, n/2-i\}/(en^2/2)$, where i is the number of 1-bits in the second half of the selected Pareto optimal solution.

Thus, the expected running time to find the Pareto front after initialization is at most $2 \cdot \sum_{i=1}^{\lceil (n-2)/4 \rceil} 2 \cdot (en^2/2)/i = O(n^2 \log n)$. By combining the expected running time $\Theta(n \log n)$ of initialization in Lemma 8.3, the expected running time of the whole process is $\Omega(n \log n)$ and $O(n^2 \log n)$. \square

A.4 Proofs in Chapter 9

Proof of Lemma 9.1.

As $F = \{J\}$ and $T = \{1, 2, \dots, m\}$, we have $|F| = 1$ and $|T| = m$. According to the construction procedure in Definition 9.2, the root node is from T with probability $m/(m+1)$, leading to $T_{init} = 1$; the root node is from F with probability $1/(m+1)$, leading to that T_{init} is the sum of the number of leaf nodes of the two subtrees, denoted by T_l and T_r , respectively. Thus,

$$\begin{aligned}\mathbb{E}[T_{init}] &= \frac{m}{m+1} \cdot 1 + \frac{1}{m+1} \cdot (\mathbb{E}[T_l] + \mathbb{E}[T_r]) \\ &= \frac{m}{m+1} + \frac{2}{m+1} \mathbb{E}[T_{init}],\end{aligned}\tag{A.54}$$

implying $\mathbb{E}[T_{init}] = m/(m-1)$;

$$\begin{aligned}\mathbb{E}[T_{init}^2] &= \frac{m}{m+1} \cdot 1^2 + \frac{1}{m+1} \cdot \mathbb{E}[(T_l + T_r)^2] \\ &\leq \frac{m}{m+1} + \frac{4}{m+1} \mathbb{E}[T_{init}^2],\end{aligned}\tag{A.55}$$

implying $\mathbb{E}[T_{init}^2] \leq m/(m-3)$. \square

Proof of Lemma 9.4.

Let $c(s)$ denote the number of connected components of a solution s . Denote s_t^* as the solution with the maximum number of nodes, i.e., the maximum $C(s)$ value, in the t -th population, i.e., the population after t iterations. We first show that $c(s_t^*)$ is non-increasing with t , i.e., $c(s_t^*) \geq c(s_{t+1}^*)$, by considering two possible cases of s_{t+1}^* . Note that the population of SMO-GP always consists of non-dominated solutions.

(1) If s_{t+1}^* is the offspring solution generated in the $(t+1)$ -th iteration, s_t^* cannot dominate s_{t+1}^* ; otherwise, s_{t+1}^* will be discarded in the updating process of SMO-GP. There are two possible situations: s_{t+1}^* weakly dominates s_t^* , or s_{t+1}^* and s_t^* are incomparable. In the first situation, $W(s_{t+1}^*) \leq W(s_t^*)$. In the second situation, s_t^* will appear in the $(t+1)$ -th population, and $C(s_{t+1}^*) > C(s_t^*)$ as s_{t+1}^* has the maximum $C(s)$ value in the $(t+1)$ -th population; thus, $W(s_{t+1}^*) < W(s_t^*)$ as they are incomparable. $W(s_{t+1}^*) \leq W(s_t^*)$ leads to $c(s_{t+1}^*) \leq c(s_t^*)$.

(2) If s_{t+1}^* is a solution in the t -th population, we show that such a case is impossible. Assume that this case holds. It implies that s_t^* must be deleted; otherwise, s_{t+1}^* cannot be the solution with the maximum number of nodes in the $(t+1)$ -th population as $C(s_t^*) > C(s_{t+1}^*)$. Based on the reproduction behavior of SMO-GP and the fact that s_t^* is deleted, it must hold that the offspring solution, denoted as s , generated in the $(t+1)$ -th iteration weakly dominates s_t^* , and s has been included into the $(t+1)$ -th population. Because s and s_{t+1}^* appear simultaneously in the $(t+1)$ -th population, they are incomparable, implying $W(s) > W(s_{t+1}^*)$ by $C(s) < C(s_{t+1}^*)$. Because s_{t+1}^* appears in the t -th population, s_{t+1}^* and s_t^* are incomparable, implying $W(s_{t+1}^*) > W(s_t^*)$ by $C(s_t^*) > C(s_{t+1}^*)$. Thus, we have $W(s) > W(s_t^*)$, contradicting with the fact that s weakly dominates s_t^* .

Next we analyze the increase of $C(s_t^*)$. The number of nodes of a solution can increase only by insertion. An offspring solution s' generated by insertion on s may be accepted only when the number of connected components decreases by 1; otherwise, $W(s') \geq W(s) \wedge C(s') > C(s)$, and then s' will be rejected. Note that in the population, every solution has a unique number of nodes, because they are non-dominated. Thus, $C(s_{t+1}^*) > C(s_t^*)$ may happen only when in the reproduction, it applies insertion on s_t^* and insertion decreases $c(s_t^*)$ by 1. Because $c(s_t^*) \leq n$ and it cannot increase with t , $c(s_t^*)$ can decrease by at most $(n-1)$ times, implying that $C(s_t^*)$ can increase by at most $(n-1)$ times.

Because insertion just increases the number of leaves by 1, the number of leaves of solutions in the population is always no larger than $T_{init} + n - 1$, implying that there are at most $(T_{init} + n)$ possible second objective, i.e., $C(s)$, values. Because there exists at most one corresponding solution for each $C(s)$ value in the population, the population size is always no larger than $T_{init} + n$. \square

A.5 Proofs in Chapter 10

Proof of Theorem 10.1.

The two EAs with/without noise are different only on whether the fitness evaluation is disturbed by noise, and thus, the number of fitness evaluations in each iteration must be the same. Therefore, comparing their expected running time is equivalent to comparing the DCFHT of their corresponding Markov chains.

In one step of the evolutionary process, denote the states before and after variation by $x \in \mathcal{X}$ and $x' \in \mathcal{X}_{var}$, respectively, and denote the state after selection by $y \in \mathcal{X}$. Because the selection process does not produce new solutions, it must hold that $y \subseteq x \cup x'$. Assume $x \in \mathcal{X}_k$ where $k \geq 1$. For $\{\xi_t\}_{t=0}^{+\infty}$, i.e., without noise, we have

$$P(\xi'_{t+1} \in \mathcal{X}_0 \mid \xi'_t = x) \leq \sum_{x' \cap \mathcal{S}^* \neq \emptyset} P_{var}(x, x'). \quad (\text{A.56})$$

For $\{\xi_t\}_{t=0}^{+\infty}$, i.e., with noise, the condition Eq. (10.8) makes that once an optimal solution is generated, it will be always accepted. Thus, we have

$$\forall k+1 \leq i \leq m :$$

$$\sum_{j=i}^m P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) \leq \sum_{\substack{x' \cap \mathcal{S}^* = \emptyset, \\ \exists y \in \cup_{j=i}^m \mathcal{X}_j : y \subseteq x \cup x'}} P_{var}(x, x'). \quad (\text{A.57})$$

By combining Eqs. (10.6) to (10.8), (A.56) and (A.57), we have

$$\forall 1 \leq i \leq m : \sum_{j=i}^m P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) \leq \sum_{j=i}^m P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x). \quad (\text{A.58})$$

Because $\sum_{j=0}^m P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) = \sum_{j=0}^m P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x) = 1$, Eq. (A.58) is equivalent to

$$\forall 0 \leq i \leq m-1 : \sum_{j=0}^i P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) \geq \sum_{j=0}^i P(\xi'_{t+1} \in \mathcal{X}_j \mid \xi'_t = x), \quad (\text{A.59})$$

implying that the condition, i.e., Eq. (7.3), of Lemma 7.1 holds. Thus, we have $\mathbb{E}[\tau \mid \xi_0 \sim \pi_0] \leq \mathbb{E}[\tau' \mid \xi'_0 \sim \pi_0]$, i.e., noise makes f easier for \mathcal{A} . \square

To prove the order of $\mathbb{E}_1(j)$ in Lemma 10.1, we need Lemmas 4.3 and A.1. Lemma 4.3 says that it is more likely that the offspring solution generated by mutating a parent solution with less 0-bits has a smaller number of 0-bits. Note that when using Lemma 4.3, we will restrict the mutation probability $p < 0.5$ rather than $p \leq 0.5$, leading to the strict inequality in the conclusion. Lemma A.1 is similar to Lemma 4.4, except that the inequalities in condition (3) and the conclusion hold strictly.

Lemma A.1. Let m ($m \geq 1$) be an integer. If it satisfies that

$$(1) \quad 0 \leq E_0 < E_1 < \cdots < E_m, \quad (\text{A.60})$$

$$(2) \quad \forall 0 \leq i \leq m : P_i, Q_i \geq 0 \wedge \sum_{i=0}^m P_i = \sum_{i=0}^m Q_i = 1, \quad (\text{A.61})$$

$$(3) \quad \forall 0 \leq k \leq m-1 : \sum_{i=0}^k P_i < \sum_{i=0}^k Q_i, \quad (\text{A.62})$$

then it holds that

$$\sum_{i=0}^m P_i \cdot E_i > \sum_{i=0}^m Q_i \cdot E_i. \quad (\text{A.63})$$

Proof. Let $R_i = P_i$ for $0 \leq i \leq m-2$, $R_{m-1} = \sum_{i=0}^{m-1} Q_i - \sum_{i=0}^{m-2} P_i$ and $R_m = Q_m$. The two vectors (R_0, \dots, R_m) and (Q_0, \dots, Q_m) satisfy conditions (2) and (3) of Lemma 4.4. Furthermore, condition (1) of Lemma 4.4, i.e., E_i increasing with i , holds. By Lemma 4.4, we have $\sum_{i=0}^m R_i \cdot E_i \geq \sum_{i=0}^m Q_i \cdot E_i$.

By comparing $\sum_{i=0}^m P_i \cdot E_i$ with $\sum_{i=0}^m R_i \cdot E_i$, we have

$$\begin{aligned} & \sum_{i=0}^m P_i \cdot E_i - \sum_{i=0}^m R_i \cdot E_i \\ &= \left(P_{m-1} - \left(\sum_{i=0}^{m-1} Q_i - \sum_{i=0}^{m-2} P_i \right) \right) E_{m-1} + (P_m - Q_m) E_m \\ &= \left(\sum_{i=0}^{m-1} P_i - \sum_{i=0}^{m-1} Q_i \right) (E_{m-1} - E_m) > 0. \end{aligned} \quad (\text{A.64})$$

Thus, $\sum_{i=0}^m P_i \cdot E_i > \sum_{i=0}^m R_i \cdot E_i \geq \sum_{i=0}^m Q_i \cdot E_i$, i.e., the lemma holds. \square

Proof of Lemma 10.1.

First, $\mathbb{E}_1(0) < \mathbb{E}_1(1)$ holds, because $\mathbb{E}_1(0) = 0$ and $\mathbb{E}_1(1) > 0$. Next, we prove $\forall 0 < j < n : \mathbb{E}_1(j) < \mathbb{E}_1(j+1)$ inductively on j .

(a) Initialization is to prove $\mathbb{E}_1(n-1) < \mathbb{E}_1(n)$. For $\mathbb{E}_1(n)$, because the next solution can be only 1^n or 0^n , $\mathbb{E}_1(n) = 1 + (1 - (1-p^n)^\lambda) \mathbb{E}_1(0) + (1-p^n)^\lambda \mathbb{E}_1(n)$, leading to $\mathbb{E}_1(n) = 1 / (1 - (1-p^n)^\lambda)$. For $\mathbb{E}_1(n-1)$, because the next solution can be 1^n , 0^n or a solution with $(n-1)$ number of 0-bits, $\mathbb{E}_1(n-1) = 1 + (1 - (1-p^{n-1}(1-p))^\lambda) \mathbb{E}_1(0) + P \cdot \mathbb{E}_1(n) + ((1-p^{n-1}(1-p))^\lambda - P) \mathbb{E}_1(n-1)$, where P denotes the probability for the next solution to be 0^n . This implies that $\mathbb{E}_1(n-1) = (1 + P \mathbb{E}_1(n)) / (1 - (1-p^{n-1}(1-p))^\lambda + P)$. Thus, we have

$$\frac{\mathbb{E}_1(n-1)}{\mathbb{E}_1(n)} = \frac{1 - (1-p^n)^\lambda + P}{1 - (1-p^{n-1}(1-p))^\lambda + P} < 1, \quad (\text{A.65})$$

where the inequality holds by $0 < p < 0.5$.

(b) Inductive Hypothesis assumes that

$$\forall K < j \leq n-1 (K \geq 1) : \mathbb{E}_1(j) < \mathbb{E}_1(j+1). \quad (\text{A.66})$$

Consider $j = K$. Let x and y be a solution with $(K+1)$ number of 0-bits and that with K number of 0-bits, respectively. Let a and b denote the number of 0-bits of the offspring solutions $\text{mut}(x)$ and $\text{mut}(y)$, respectively; that is, $a = |\text{mut}(x)|_0$ and $b = |\text{mut}(y)|_0$. For the λ independent mutations on x and y , we use a_1, \dots, a_λ and b_1, \dots, b_λ , respectively. Note that a_1, \dots, a_λ are i.i.d., and b_1, \dots, b_λ are also i.i.d. Let $p_j = P(a_i \leq j)$ and $q_j = P(b_i \leq j)$. By Lemma 4.3, we have $\forall 0 \leq j \leq n-1 : p_j < q_j$.

For $\mathbb{E}_1(K+1)$, let P_0 and P_i ($1 \leq i \leq n$) be the probability that for the λ offspring solutions, the smallest number of 0-bits is 0, and the largest number of 0-bits is i while the smallest number of 0-bits is larger than 0, respectively. In other words, $P_0 = P(\min\{a_1, \dots, a_\lambda\} = 0)$ and $P_i = P(\max\{a_1, \dots, a_\lambda\} = i \wedge \min\{a_1, \dots, a_\lambda\} > 0)$. Considering the mutation and selection behavior of $(1+\lambda)$ -EA on the Trap problem, we have

$$\mathbb{E}_1(K+1) = 1 + P_0 \mathbb{E}_1(0) + \sum_{i=1}^{K+1} P_i \mathbb{E}_1(K+1) + \sum_{i=K+2}^n P_i \mathbb{E}_1(i). \quad (\text{A.67})$$

For $\mathbb{E}_1(K)$, let $Q_0 = P(\min\{b_1, \dots, b_\lambda\} = 0)$ and $Q_i = P(\max\{b_1, \dots, b_\lambda\} = i \wedge \min\{b_1, \dots, b_\lambda\} > 0)$. We have

$$\mathbb{E}_1(K) = 1 + Q_0 \mathbb{E}_1(0) + \sum_{i=1}^K Q_i \mathbb{E}_1(K) + \sum_{i=K+1}^n Q_i \mathbb{E}_1(i). \quad (\text{A.68})$$

To compare $\mathbb{E}_1(K+1)$ with $\mathbb{E}_1(K)$, we show that

$$\forall 0 \leq j \leq n-1 : \sum_{i=0}^j P_i < \sum_{i=0}^j Q_i. \quad (\text{A.69})$$

For $\sum_{i=0}^j P_i$, we have

$$\begin{aligned} \sum_{i=0}^j P_i &= P(\min\{a_1, \dots, a_\lambda\} = 0) \\ &\quad + P(\max\{a_1, \dots, a_\lambda\} \leq j \wedge \min\{a_1, \dots, a_\lambda\} > 0) \\ &= P(a_1 = 0 \vee \dots \vee a_\lambda = 0) + P(0 < a_1 \leq j \wedge \dots \wedge 0 < a_\lambda \leq j) \\ &= 1 - (1 - p_0)^\lambda + (p_j - p_0)^\lambda \\ &< 1 - (1 - p_0)^\lambda + (q_j - p_0)^\lambda, \end{aligned} \quad (\text{A.70})$$

where Eq. (A.70) holds by $p_j < q_j$. For $\sum_{i=0}^j Q_i$, we similarly have $\sum_{i=0}^j Q_i = 1 - (1 - q_0)^\lambda + (q_j - q_0)^\lambda$. Thus,

$$\begin{aligned}
\sum_{i=0}^j Q_i - \sum_{i=0}^j P_i &> (1-p_0)^\lambda - (1-q_0)^\lambda + (q_j - q_0)^\lambda - (q_j - p_0)^\lambda \\
&= ((1-q_0 + q_0 - p_0)^\lambda - (1-q_0)^\lambda) - ((q_j - q_0 + q_0 - p_0)^\lambda - (q_j - q_0)^\lambda) \\
&= f(1-q_0) - f(q_j - q_0),
\end{aligned} \tag{A.71}$$

where Eq. (A.71) holds by letting $f(x) = (x+q_0-p_0)^\lambda - x^\lambda$. As $q_0 > p_0$, it can be verified that $f(x)$ is increasing. By $q_j < 1$, we have $f(1-q_0) > f(q_j - q_0)$, implying that Eq. (A.69) holds.

By subtracting $\mathbb{E}_1(K)$ from $\mathbb{E}_1(K+1)$, we have

$$\begin{aligned}
\mathbb{E}_1(K+1) - \mathbb{E}_1(K) &= \left(P_0 \mathbb{E}_1(0) + \sum_{i=1}^{K+1} P_i \mathbb{E}_1(K+1) + \sum_{i=K+2}^n P_i \mathbb{E}_1(i) \right. \\
&\quad \left. - Q_0 \mathbb{E}_1(0) - \sum_{i=1}^{K+1} Q_i \mathbb{E}_1(K+1) - \sum_{i=K+2}^n Q_i \mathbb{E}_1(i) \right) \\
&\quad + \sum_{i=1}^K Q_i (\mathbb{E}_1(K+1) - \mathbb{E}_1(K)) \\
&> \sum_{i=1}^K Q_i (\mathbb{E}_1(K+1) - \mathbb{E}_1(K)),
\end{aligned} \tag{A.72}$$

where Eq. (A.72) holds by applying Lemma A.1 to the formula in big brackets. The three conditions of Lemma A.1 can be verified, because $\mathbb{E}_1(0) = 0 < \mathbb{E}_1(K+1) < \dots < \mathbb{E}_1(n)$ by inductive hypothesis; $\sum_{i=0}^n P_i = \sum_{i=0}^n Q_i = 1$; Eq. (A.69) holds. As $\sum_{i=1}^K Q_i < 1$, we have $\mathbb{E}_1(K+1) > \mathbb{E}_1(K)$.

(c) **Concluding** from (a) and (b), the lemma holds. \square

Proof of Theorem 10.3.

Let $\{\xi_t\}_{t=0}^{+\infty}/\{\xi'_t\}_{t=0}^{+\infty}$ model (1+1)-EA \neq with/without one-bit noise for maximizing the Peak problem. Note that one state x corresponds to one solution $s \in \{0, 1\}^n$ here. Both the CFHT $\mathbb{E}[\tau \mid \xi_0 = x]$ and $\mathbb{E}[\tau' \mid \xi'_0 = x]$ depend only on $|x|_0$. Let $\mathbb{E}(i)$ and $\mathbb{E}'(i)$ denote $\mathbb{E}[\tau \mid \xi_0 = x]$ and $\mathbb{E}[\tau' \mid \xi'_0 = x]$ with $|x|_0 = i$, respectively.

For $\{\xi'_t\}_{t=0}^{+\infty}$, i.e., without noise, starting from a solution x with $|x|_0 = i > 0$, in one step, any non-optimal offspring solution has the same fitness as the parent solution and will be rejected due to the strict selection strategy; only the optimal solution can be accepted, occurring with probability $(1/n^i)(1 - 1/n)^{n-i}$. Thus, we have

$$\mathbb{E}'(i) = 1 + \frac{1}{n^i} \left(1 - \frac{1}{n} \right)^{n-i} \mathbb{E}'(0) + \left(1 - \frac{1}{n^i} \left(1 - \frac{1}{n} \right)^{n-i} \right) \mathbb{E}'(i), \tag{A.73}$$

leading to $\mathbb{E}'(i) = n^i(n/(n-1))^{n-i}$.

For $\{\xi_t\}_{t=0}^{+\infty}$, i.e., with one-bit noise, assume using re-evaluation, which re-evaluates the fitness of the parent solution and evaluates the fitness of the offspring solution in each iteration of Algorithm 2.1. When starting from x with $|x|_0 = 1$, if the generated offspring solution x' is the optimal solution 1^n , it will be accepted with probability $(1 - p_n)(1 - p_n/n)$ because only no bit flip for noise on x' and no 0-bit flip for noise on x can make $f^n(x') > f^n(x)$; otherwise, $|x|_0 = 1$ will be kept, because $f^n(x') \leq f^n(x) \forall x' \text{ with } |x'|_0 \geq 2$. Thus, we have

$$\begin{aligned}\mathbb{E}(1) = & 1 + \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} (1 - p_n) \left(1 - \frac{p_n}{n}\right) \mathbb{E}(0) \\ & + \left(1 - \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} (1 - p_n) \left(1 - \frac{p_n}{n}\right)\right) \mathbb{E}(1),\end{aligned}\quad (\text{A.74})$$

leading to

$$\mathbb{E}(1) = n \left(\frac{n}{n-1}\right)^{n-1} \frac{1}{(1 - p_n)(1 - \frac{p_n}{n})}. \quad (\text{A.75})$$

When starting from x with $|x|_0 = i \geq 2$, if the offspring solution x' is 1^n , it will be accepted with probability $(1 - p_n)$ because only no bit flip for noise on x' can make $f^n(x') > f^n(x)$; if $|x'|_0 = 1$, it will be accepted with probability p_n/n because only flipping the unique 0-bit for noise on x' can make $f^n(x') > f^n(x)$; otherwise, $|x|_0 = i$ will be kept, because $f^n(x') = f^n(x) \forall x'$ with $|x'|_0 \geq 2$. Let $mut_{i \rightarrow 1}$ be the probability of mutating $|x|_0 = i$ to $|x'|_0 = 1$ by bit-wise mutation with the mutation probability $p = 1/n$. For $i \geq 2$,

$$\begin{aligned}\mathbb{E}(i) = & 1 + \frac{(n-1)^{n-i}}{n^n} (1 - p_n) \mathbb{E}(0) + mut_{i \rightarrow 1} \cdot \frac{p_n}{n} \mathbb{E}(1) \\ & + \left(1 - \frac{(n-1)^{n-i}}{n^n} (1 - p_n) - mut_{i \rightarrow 1} \frac{p_n}{n}\right) \mathbb{E}(i),\end{aligned}\quad (\text{A.76})$$

leading to

$$\mathbb{E}(i) = \frac{1 + mut_{i \rightarrow 1} \frac{p_n}{n} \mathbb{E}(1)}{\frac{1}{n^i} (1 - \frac{1}{n})^{n-i} (1 - p_n) + mut_{i \rightarrow 1} \frac{p_n}{n}}. \quad (\text{A.77})$$

By using re-evaluation under noise, each iteration of (1+1)-EA $^\neq$ requires two fitness evaluations. If without noise, (1+1)-EA $^\neq$ evaluates only the offspring solution in each iteration. Thus, its expected running time without/with one-bit noise is $(1 + \mathbb{E}'(i))/(1 + 2\mathbb{E}(i))$, where the term 1 corresponds to one fitness evaluation of the initial solution. To prove that one-bit noise can be helpful, we need to show that $\exists i \geq 1 : 2\mathbb{E}(i) < \mathbb{E}'(i)$. It is clear that $i = 1$ is impossible because $\mathbb{E}(1) > \mathbb{E}'(1)$. For larger i with $i > (1 + p_n)/(p_n(1 - 1/n))$, we have

$$\begin{aligned}
& (2\mathbb{E}(i) - \mathbb{E}'(i)) \cdot \left(\frac{1}{n^i} \left(1 - \frac{1}{n} \right)^{n-i} (1 - p_n) + mut_{i \rightarrow 1} \frac{p_n}{n} \right) \\
&= 1 + p_n - mut_{i \rightarrow 1} \frac{p_n}{n} \left(n^i \left(\frac{n}{n-1} \right)^{n-i} - 2\mathbb{E}(1) \right) \\
&\leq 1 + p_n - \frac{i}{n^{i-1}} \left(1 - \frac{1}{n} \right)^{n-i+1} \frac{p_n}{n} n^i \left(\frac{n}{n-1} \right)^{n-i} \tag{A.78}
\end{aligned}$$

$$= 1 + p_n - ip_n \left(1 - \frac{1}{n} \right) < 0, \tag{A.79}$$

where Eq. (A.78) holds because $mut_{i \rightarrow 1} \geq (i/n^{i-1})(1-1/n)^{n-i+1}$ and $\mathbb{E}(1) \ll n^i(n/(n-1))^{n-i}$ for large enough n and p_n being constant, and Eq. (A.79) holds by $i > (1+p_n)/(p_n(1-1/n))$. Eq. (A.79) is equivalent to $2\mathbb{E}(i) - \mathbb{E}'(i) < 0$, implying that noise is helpful when starting from an initial solution x with $|x|_0 > (1+p_n)/(p_n(1-1/n))$. \square

Proof of Lemma 10.2.

We prove $\forall 0 \leq j < n : \mathbb{E}_2(j) < \mathbb{E}_2(j+1)$ inductively on j .

(a) Initialization is to prove $\mathbb{E}_2(0) < \mathbb{E}_2(1)$, which holds because $\mathbb{E}_2(1) > 0 = \mathbb{E}_2(0)$.

(b) Inductive Hypothesis assumes that

$$\forall 0 \leq j < K (K \leq n-1) : \mathbb{E}_2(j) < \mathbb{E}_2(j+1). \tag{A.80}$$

Consider $j = K$. We use the similar analysis method as that in the proof of Lemma 10.1 to compare $\mathbb{E}_2(K+1)$ with $\mathbb{E}_2(K)$.

For $\mathbb{E}_2(K+1)$, let P_i ($0 \leq i \leq n$) be the probability for the smallest number of 0-bits of the λ offspring solutions to be i ; that is, $P_i = P(\min\{a_1, \dots, a_\lambda\} = i)$. Considering the mutation and selection behavior of $(1+\lambda)$ -EA on the OneMax problem, we have

$$\mathbb{E}_2(K+1) = \sum_{i=0}^K P_i \mathbb{E}_2(i) + \sum_{i=K+1}^n P_i \mathbb{E}_2(K+1). \tag{A.81}$$

For $\mathbb{E}_2(K)$, let $Q_i = P(\min\{b_1, \dots, b_\lambda\} = i)$. We have

$$\mathbb{E}_2(K) = \sum_{i=0}^{K-1} Q_i \mathbb{E}_2(i) + \sum_{i=K}^n Q_i \mathbb{E}_2(K). \tag{A.82}$$

By subtracting $\mathbb{E}_2(K)$ from $\mathbb{E}_2(K+1)$, we have

$$\mathbb{E}_2(K+1) - \mathbb{E}_2(K) = \sum_{i=K+1}^n P_i (\mathbb{E}_2(K+1) - \mathbb{E}_2(K)) +$$

$$\begin{aligned} & \left(\sum_{i=0}^{K-1} P_i \mathbb{E}_2(i) + \sum_{i=K}^n P_i \mathbb{E}_2(K) - \sum_{i=0}^{K-1} Q_i \mathbb{E}_2(i) - \sum_{i=K}^n Q_i \mathbb{E}_2(K) \right) \\ & > \sum_{i=K+1}^n P_i (\mathbb{E}_2(K+1) - \mathbb{E}_2(K)), \end{aligned} \quad (\text{A.83})$$

where Eq. (A.83) holds by applying Lemma A.1 to the formula in big brackets. The conditions of Lemma A.1 can be verified, because $\mathbb{E}_2(0) < \dots < \mathbb{E}_2(K)$ by inductive hypothesis; $\sum_{i=0}^n P_i = \sum_{i=0}^n Q_i = 1$; Eq. (A.84) holds.

$$\begin{aligned} \sum_{i=0}^j Q_i - \sum_{i=0}^j P_i &= P(\min\{b_1, \dots, b_\lambda\} \leq j) - P(\min\{a_1, \dots, a_\lambda\} \leq j) \\ &= P(b_1 \leq j \vee \dots \vee b_\lambda \leq j) - P(a_1 \leq j \vee \dots \vee a_\lambda \leq j) \\ &= 1 - (1 - q_j)^\lambda - (1 - (1 - p_j)^\lambda) > 0, \end{aligned} \quad (\text{A.84})$$

where Eq. (A.84) holds by $p_j < q_j$. Because $\sum_{i=K+1}^n P_i < 1$, Eq. (A.83) implies $\mathbb{E}_2(K+1) > \mathbb{E}_2(K)$.

(c) Concluding from (a) and (b), the lemma holds. \square

Proof of Lemma 10.4.

Let i denote the number of 0-bits of the current solution x , where $0 \leq i \leq n$. Here, an offspring solution x' will be accepted only if $f^n(x') - f^n(x) \geq 2$. As in the proof of Lemma 10.3, we can derive

$$\forall d \geq 1 : p_{i,i+d} = 0, \quad (\text{A.85})$$

$$\forall d \geq 2 : p_{i,i-d} > 0, \quad (\text{A.86})$$

$$p_{i,i-1} = P_{-1} \left(p_n \frac{n-i}{n} \left(1 - p_n + p_n \frac{i-1}{n} \right) + (1 - p_n) \left(p_n \frac{i-1}{n} \right) \right). \quad (\text{A.87})$$

Thus, i will never increase and can decrease in one iteration with probability at least

$$\begin{aligned} p_{i,i-1} &\geq \frac{i}{n} \left(1 - \frac{1}{n} \right)^{n-1} \left((1 - p_n)p_n \left(1 - \frac{1}{n} \right) + p_n^2 \frac{(n-i)(i-1)}{n^2} \right) \\ &\geq \frac{i(1 - p_n)p_n}{2en}, \end{aligned} \quad (\text{A.88})$$

implying that the expected number of iterations until $i = 0$, i.e., the optimal solution is found, is at most

$$\sum_{i=1}^n \frac{2en}{i(1 - p_n)p_n} = O \left(\frac{n \log n}{p_n(1 - p_n)} \right). \quad (\text{A.89})$$

\square

Proof of Lemma 10.5.

Assume that the number of 1-bits of the initial solution x is less than $n - 1$, i.e., $|x|_1 < n - 1$. Let T denote the running time until finding the optimal solution 1^n when starting from x . Denote A as the event that in the evolutionary process, any solution x' with $|x'|_1 = n - 1$ is never found. By the law of total expectation, we have

$$\mathbb{E}[T] = \mathbb{E}[T | A] \cdot P(A) + \mathbb{E}[T | \bar{A}] \cdot P(\bar{A}). \quad (\text{A.90})$$

For any evolutionary path with the event A happening, it has to flip at least two bits in the last step for finding the optimal solution, because any solution x' with $|x'|_1 = n - 1$ is never found. Thus, we have $P(A) \leq \binom{n}{2} (1/n^2) \leq 1/2$. As $P(\bar{A}) + P(A) = 1$, we have $P(\bar{A}) \geq 1/2$, implying

$$\mathbb{E}[T] \geq \mathbb{E}[T | \bar{A}] \cdot P(\bar{A}) \geq \frac{1}{2} \cdot \mathbb{E}[T | \bar{A}]. \quad (\text{A.91})$$

We then derive a lower bound of $\mathbb{E}[T | \bar{A}]$. We divide the running time T into two parts: the running time until finding a solution x' with $|x'|_1 = n - 1$ for the first time, denoted by T_1 , and the remaining running time for finding the optimal solution, denoted by T_2 . Thus,

$$\mathbb{E}[T | \bar{A}] = \mathbb{E}[T_1 | \bar{A}] + \mathbb{E}[T_2 | \bar{A}]. \quad (\text{A.92})$$

According to the upper bound analysis in the proof of Lemma 10.4, once a solution x with $|x|_0 = 1$ is found, it will always satisfy $|x|_0 = 1$ before finding the optimal solution, because $\forall d \geq 1 : p_{i,i+d} = 0$. Meanwhile, the optimal solution will be found by one step with probability $p_{1,0} = (1/n)(1 - 1/n)^{n-1} p_n (1 - p_n) (1 - 1/n) \leq p_n (1 - p_n) / (en)$. Thus, we have

$$\mathbb{E}[T_2 | \bar{A}] \geq \frac{en}{p_n (1 - p_n)}, \quad (\text{A.93})$$

implying that $\mathbb{E}[T | \bar{A}] \geq en / (p_n (1 - p_n))$, and thus $\mathbb{E}[T] \geq en / (2p_n (1 - p_n))$.

Because the initial solution is uniformly distributed over $\{0, 1\}^n$, we have $P(|x|_1 < n - 1) = 1 - (n + 1)/2^n$. Thus, the expected running time of the whole process is lower bounded by $(1 - (n + 1)/2^n) \cdot (en) / (2p_n (1 - p_n)) = \Omega(n / (p_n (1 - p_n)))$.

We derive another lower bound which does not depend on p_n . From Lemma 10 in [Droste et al., 2002], the expected running time of (1+1)-EA to optimize linear functions with positive weights is $\Omega(n \log n)$. Their proof idea is to analyze the expected running time until all 0-bits of the initial solution have been flipped at least once, which is a lower bound of the expected running time until finding the optimal solution 1^n . Because noise will not affect this analytic process, we can apply their result directly to our setting, and derive the lower bound $\Omega(n \log n)$.

By combining the derived two lower bounds, the expected running time is lower bounded by $\Omega(n \log n + n / (p_n (1 - p_n)))$. \square

Proof of Theorem 10.8.

Let $i = |\mathbf{x}|_0$ for the current solution \mathbf{x} . An offspring solution \mathbf{x}' will be accepted only if $f^n(\mathbf{x}') - f^n(\mathbf{x}) \geq \tau > 2$. Thus, we have

$$\forall d \geq 1 : p_{i,i+d} = 0, \quad (\text{A.94})$$

$$p_{i,i-1} = \begin{cases} P_{-1} \cdot (p_n \frac{n-i}{n} p_n \frac{i-1}{n}) & \text{if } \tau = 3; \\ 0 & \text{otherwise.} \end{cases} \quad (\text{A.95})$$

In the evolutionary process, there exists some positive probability of finding a solution \mathbf{x} with $|\mathbf{x}|_0 = 1$, i.e., $i = 1$. Once it happens, $i = 1$ will always hold because $p_{1,0} = 0$ and $\forall d \geq 1 : p_{1,1+d} = 0$. In such a situation, the optimal solution 1^n will never be found. Thus, the expected running time is infinite. \square

Proof of Theorem 10.9.

We analyze the expected running time for each value of τ , respectively.

For $\tau = 0$, we use the simplified negative drift analysis approach in Theorem 2.5 to prove an exponential running time lower bound. Let x_t be the number of 0-bits of the solution after t iterations of (1+1)-EA. We consider the interval $[0, n^{1/4}]$, i.e., the parameters $a = 0$ and $b = n^{1/4}$ in Theorem 2.5, and analyze $\mathbb{E}[x_t - x_{t+1} \mid x_t = i]$ for $1 \leq i < n^{1/4}$. Let $p_{i,i+d}$ denote the probability for the next solution after bit-wise mutation and selection to have $(i+d)$ number of 0-bits, i.e., $x_{t+1} = i+d$, where $-i \leq d \leq n-i$. We have

$$\mathbb{E}[x_t - x_{t+1} \mid x_t = i] = \sum_{d=1}^i d \cdot p_{i,i-d} - \sum_{d=1}^{n-i} d \cdot p_{i,i+d}. \quad (\text{A.96})$$

Let P_d denote the probability for the offspring solution generated by bit-wise mutation to have $(i+d)$ number of 0-bits. Using the analytic procedure of $p_{i,i+d}$ in the proof of Lemma 10.3 and considering $p_n = 1$ and $\tau = 0$, we have $\forall d \geq 3 : p_{i,i+d} = 0$; $p_{i,i+2} = P_2/4$; $p_{i,i+1} = P_1/4$; $p_{i,i-1} \leq 3P_{-1}/4$; $\forall 2 \leq d \leq i : p_{i,i-d} = P_{-d}$. Because it is sufficient to flip one 1-bit and keep the other bits unchanged, $P_1 \geq ((n-i)/n)(1 - 1/n)^{n-1} \geq (n-i)/(en)$. Because it is necessary to flip at least d number of 0-bits, $P_{-d} \leq \binom{i}{d}(1/n^d)$. By applying these probabilities to Eq. (A.96), we have

$$\begin{aligned} \mathbb{E}[x_t - x_{t+1} \mid x_t = i] &\leq \frac{3P_{-1}}{4} + \sum_{d=2}^i d \cdot P_{-d} - \frac{P_1}{4} \\ &\leq \frac{3i}{4n} + \sum_{d=2}^i d \cdot \binom{i}{d} \frac{1}{n^d} - \frac{n-i}{4en} = \frac{i}{n} \left(\left(1 + \frac{1}{n}\right)^{i-1} + \frac{1}{4e} - \frac{1}{4} \right) - \frac{1}{4e} \\ &= -\frac{1}{4e} + O\left(\frac{n^{1/4}}{n}\right), \end{aligned} \quad (\text{A.97})$$

where Eq. (A.97) holds by $i < n^{1/4}$. Thus, $\mathbb{E}[x_t - x_{t+1} \mid x_t = i] = -\Omega(1)$, implying that condition (1) of Theorem 2.5 holds. For condition (2), it requires to examine $P(|x_{t+1} - x_t| \geq j \mid x_t \geq 1)$. Because it is necessary to flip at least j bits, we have

$$P(|x_{t+1} - x_t| \geq j \mid x_t \geq 1) \leq \binom{n}{j} \frac{1}{n^j} \leq \frac{1}{j!} \leq 2 \cdot \frac{1}{2^j}, \quad (\text{A.98})$$

implying that condition (2) of Theorem 2.5 holds with $\delta = 1$ and $r(l) = 2$. Note that $l = b - a = n^{1/4}$. Thus, by Theorem 2.5, the probability that the running time is $2^{O(n^{1/4})}$, is exponentially small when starting from a solution s with $|s|_0 \geq n^{1/4}$. Due to the uniform initial distribution, the probability that the initial solution s has $|s|_0 < n^{1/4}$ is exponentially small by the Chernoff bound. Thus, the expected running time is exponential.

For $\tau = 1$, we use the same analytic procedure as $\tau = 0$. The only difference is the calculation of $p_{i,i+d}$: $\forall d \geq 2 : p_{i,i+d} = 0$; $p_{i,i+1} = P_1/4$; $p_{i,i-1} \leq 3P_{-1}/4$; $p_{i,i-2} \leq 3P_{-2}/4$; $\forall 3 \leq d \leq i : p_{i,i-d} = P_{-d}$. The analysis of the two conditions of Theorem 2.5 will not be affected. Thus, we derive the same result as $\tau = 0$: the expected running time is exponential.

For $\tau \geq 2$, we have $\forall d \geq 1 : p_{i,i+d} = 0$; $\forall 2 \leq d \leq i : p_{i,i-d} > 0$. For $p_{i,i-1}$, we consider two cases: $\forall i \geq 2 : p_{i,i-1} = P_{-1}/4$; $p_{1,0} = 0$. There exists some positive probability of finding a solution s with $|s|_0 = 1$. For $\tau \geq 2$, we have $\forall d \geq 1 : p_{1,1+d} = 0$ and $p_{1,0} = 0$. Thus, it will always be kept in such a state, i.e., $|s|_0 = 1$, implying that the expected running time for finding the optimal solution 1^n is infinite. \square

To prove Theorem 10.10, we will view the evolutionary process as a random walk on a graph in Algorithm A.1, which has often been used for analyzing randomized search heuristics, e.g., in [Giel and Wegener, 2003, Neumann and Witt, 2010].

Algorithm A.1 Random Walk

Input: undirected connected graph $G = (V, E)$ with vertex set V and edge set E

Process:

- 1: start at a vertex $v \in V$;
 - 2: **while** criterion is not met **do**
 - 3: choose a neighbor u of v uniformly at random;
 - 4: set $v = u$
 - 5: **end while**
-

Lemma A.2 gives an upper bound of the expected number of steps for a random walk to visit each vertex of a graph at least once.

Lemma A.2. [Aleliunas et al., 1979] *For a random walk on an undirected connected graph $G = (V, E)$, the expected number of steps until each vertex $v \in V$ has been visited at least once is upper bounded by $2|E|(|V| - 1)$.*

Proof of Theorem 10.10.

Let i denote the number of 0-bits of the current solution x , where $0 \leq i \leq n$. We first analyze $p_{i,i+d}$ as in the proof of Lemma 10.3. There are differences in the analysis, caused by different threshold and noise settings, respectively. Due to the threshold difference, the acceptance probability is different when the fitness gap between the offspring solution x' and the parent solution x is 1: x' will be accepted with probability $1/(2en)$, whereas it will be always accepted in the proof of Lemma 10.3. Due to the noise difference, the probability of flipping a 0 or 1-bit is different: a random 0 or 1-bit will be flipped with an equal probability of $1/2$, whereas a uniformly randomly chosen bit will be flipped in the proof of Lemma 10.3.

Similarly, we derive the value of $p_{i,i+d}$ for $1 \leq i \leq n-1$. It can be seen that $\forall d \geq 2 : p_{i,i+d} = 0 \wedge p_{i,i-d} > 0$. For $p_{i,i+1}$, the offspring solution x' will be accepted only if $f^n(x') = n-i \wedge f^n(x) = n-i-1$, and the acceptance probability is $1/(2en)$. The probability of $f^n(x') = n-i$ is at most p_n , as it requires to flip one 0-bit of x' in noise; the probability of $f^n(x) = n-i-1$ is $p_n(1/2)$, as it requires to flip one 1-bit of x . Thus, $p_{i,i+1} \leq P_1(p_n(1/2) \cdot p_n) \cdot (1/(2en))$. For $p_{i,i-1}$, we consider two cases:

(1) $2 \leq i \leq n-1$. If $f^n(x) = n-i-1$, occurring with probability $p_n(1/2)$, there are three cases for the offspring solution x' : If $f^n(x') = n-i$, occurring with probability $p_n(1/2)$, the acceptance probability is $1/(2en)$, as $f^n(x') = f^n(x) + 1$; if $f^n(x') = n-i+1$ or $n-i+2$, occurring with probability $(1-p_n) + p_n(1/2)$, the acceptance probability is 1, as $f^n(x') > f^n(x) + 1$. If $f^n(x) = n-i$, occurring with probability $1-p_n$, there are two cases for the acceptance of x' : If $f^n(x') = n-i+1$, occurring with probability $1-p_n$, the acceptance probability is $1/(2en)$; if $f^n(x') = n-i+2$, occurring with probability $p_n(1/2)$, the acceptance probability is 1. If $f^n(x) = n-i+1$, occurring with probability $p_n(1/2)$, x' will be accepted only if $f^n(x') = n-i+2$, occurring with probability $p_n(1/2)$, and the acceptance probability is $1/(2en)$. Thus, we have

$$\begin{aligned} p_{i,i-1} = P_{-1} &\left(p_n \frac{1}{2} \left(p_n \frac{1}{2} \cdot \frac{1}{2en} + (1-p_n) + p_n \frac{1}{2} \right) \right. \\ &\left. + (1-p_n) \left((1-p_n) \cdot \frac{1}{2en} + p_n \frac{1}{2} \right) + p_n \frac{1}{2} p_n \frac{1}{2} \cdot \frac{1}{2en} \right). \end{aligned} \quad (\text{A.99})$$

(2) $i=1$. The analysis is similar to case (1). The only difference is that when noise happens, $f^n(x') = n-i$ with probability 1 here because $|x'|_0 = i-1 = 0$ reaches the extreme case, whereas in case (1), $f^n(x') = n-i$ or $n-i+2$ with an equal probability of $1/2$. Considering $P_{-1} = (1/n)(1-1/n)^{n-1}$ for $i=1$, we have

$$p_{1,0} = \frac{1}{n} \left(1 - \frac{1}{n} \right)^{n-1} \cdot \left(p_n \frac{1}{2} \left(p_n \frac{1}{2} \cdot \frac{1}{2en} + (1-p_n) \right) + (1-p_n)(1-p_n) \frac{1}{2en} \right). \quad (\text{A.100})$$

Our goal is to reach $i = 0$. Starting from $i = 1$, i will reach 0 in one step with probability

$$p_{1,0} \geq \frac{1}{en} \cdot \frac{1}{2en} \cdot \left(\frac{p_n^2}{2} + (1 - p_n)^2 \right) \geq \frac{1}{6e^2 n^2}, \quad (\text{A.101})$$

where the last inequality holds by $0 \leq p_n \leq 1$. Thus, for reaching $i = 0$, it requires to reach $i = 1$ for $O(n^2)$ times in expectation.

Next, we analyze the expected running time until $i = 1$. In this process, we can pessimistically assume that $i = 0$ will never be reached, because our final goal is to derive a running time upper bound for reaching $i = 0$. For $2 \leq i \leq n - 1$, we have

$$\begin{aligned} \frac{p_{i,i-1}}{p_{i,i+1}} &\geq \frac{P_{-1} \cdot (p_n \frac{1}{2} p_n \frac{1}{2})}{P_1 \cdot (p_n \frac{1}{2} p_n) \cdot \frac{1}{2en}} \\ &\geq \frac{\frac{i}{n} (1 - \frac{1}{n})^{n-1} \cdot (p_n \frac{1}{2} p_n \frac{1}{2})}{\frac{n-i}{n} \cdot (p_n \frac{1}{2} p_n) \cdot \frac{1}{2en}} \geq \frac{n}{n-i} > 1. \end{aligned} \quad (\text{A.102})$$

Again, we can pessimistically assume that $p_{i,i-1} = p_{i,i+1}$ and $\forall d \geq 2 : p_{i,i-d} = 0$, because the goal is to derive an upper bound of the expected running time until $i = 1$. We can view the evolutionary process for reaching $i = 1$ as a random walk on the path $\{1, 2, \dots, n-1, n\}$. We call a step that jumps to the neighbor state a relevant step. By Lemma A.2, it requires at most $2(n-1)^2$ expected number of relevant steps to reach $i = 1$. Because the probability of a relevant step is at least

$$p_{i,i-1} \geq \frac{i}{en} \cdot \frac{1}{2en} \left((1 - p_n)^2 + p_n^2 \frac{1}{2} \right) \geq \frac{2}{2e^2 n^2} \cdot \frac{1}{3}, \quad (\text{A.103})$$

the expected running time for a relevant step is $O(n^2)$. Thus, the expected running time for reaching $i = 1$ is $O(n^4)$, implying that the expected running time of the whole optimization process is $O(n^6)$. \square

Proof of Theorem 10.11.

We use Theorem 2.6 for the proof. Let x_t be the number of 0-bits of the solution after t iterations of (1+1)-EA. We consider the interval $[0, n^{1/4}]$, i.e., the parameters $a = 0$ and $b = n^{1/4}$ in Theorem 2.6.

We analyze $\mathbb{E}[x_t - x_{t+1} | x_t = i]$ for $1 \leq i < n^{1/4}$. Let $p_{i,i+d}$ denote the probability for the next solution after bit-wise mutation and selection to have $(i+d)$ number of 0-bits, i.e., $x_{t+1} = i+d$, where $-i \leq d \leq n-i$. Thus,

$$\mathbb{E}[x_t - x_{t+1} | x_t = i] = \sum_{d=1}^i d \cdot p_{i,i-d} - \sum_{d=1}^{n-i} d \cdot p_{i,i+d}. \quad (\text{A.104})$$

Next we analyze the probabilities $p_{i,i+d}$ for $i \geq 1$. Let P_d denote the probability for the offspring solution s' generated by bit-wise mutation to

have $(i + d)$ number of 0-bits. Note that one-bit noise with $p_n = 1$ makes the noisy fitness and the true fitness of a solution have a gap of one, i.e., $|f^n(s) - f(s)| = 1$. For a solution s with $|s|_0 = i$, $f^n(s) = n - i + 1$ with a probability of i/n ; otherwise, $f^n(s) = n - i - 1$. Let s and s' denote the current solution and the offspring solution, respectively.

- (1) When $d \geq 3$, $f^n(s') \leq n - i - d + 1 \leq n - i - 2 < f^n(s)$. Thus, the offspring solution s' will be discarded in this case, implying $\forall d \geq 3 : p_{i,i+d} = 0$.
- (2) When $d = 2$, the offspring solution s' will be accepted iff $f^n(s') = n - i - 1 = f^n(s)$, the probability of which is $((i+2)/n) \cdot ((n-i)/n)$, as it requires to flip one 0-bit of s' and one 1-bit of s in noisy evaluation. Thus, $p_{i,i+2} = P_2 \cdot ((i+2)/n)((n-i)/n)$.
- (3) When $d = 1$, s' will be accepted iff $f^n(s') = n - i \wedge f^n(s) = n - i - 1$, the probability of which is $((i+1)/n) \cdot ((n-i)/n)$, as it requires to flip one 0-bit of s' and one 1-bit of s in noisy evaluation. Thus, $p_{i,i+1} = P_1 \cdot ((i+1)/n)((n-i)/n)$.
- (4) When $d = -1$, s' will be rejected iff $f^n(s') = n - i \wedge f^n(s) = n - i + 1$, the probability of which is $((n-i+1)/n) \cdot (i/n)$, as it requires to flip one 1-bit of s' and one 0-bit of s in noisy evaluation. Thus, $p_{i,i-1} = P_{-1} \cdot (1 - ((n-i+1)/n)(i/n))$.
- (5) When $d \leq -2$, $f^n(s') \geq n - i - d - 1 \geq n - i + 1 \geq f^n(s)$. Thus, the offspring solution s' will always be accepted in this case, implying $\forall d \leq -2 : p_{i,i+d} = P_d$.

We then bound the probabilities P_d . For $d > 0$, $P_d \geq \binom{n-i}{d} (1/n^d) (1 - 1/n)^{n-d}$, because it is sufficient to flip d 1-bits and keep the other bits unchanged; $P_{-d} \leq \binom{i}{d} (1/n^d)$, because it is necessary to flip at least d 0-bits. Thus, we can upper bound $\sum_{d=2}^i dP_{-d}$ as follows:

$$\begin{aligned} \sum_{d=2}^i dP_{-d} &\leq \sum_{d=2}^i d \binom{i}{d} \frac{1}{n^d} = \sum_{d=1}^i d \binom{i}{d} \frac{1}{n^d} - \frac{i}{n} \\ &= \frac{i}{n} \sum_{d=0}^{i-1} \binom{i-1}{d} \frac{1}{n^d} - \frac{i}{n} = \frac{i}{n} \left(\left(1 + \frac{1}{n} \right)^{i-1} - 1 \right). \end{aligned} \quad (\text{A.105})$$

For the probability P_{-1} , we also need a tighter upper bound from Lemma 2 in [Paixão et al., 2015], i.e.,

$$P_{-1} \leq \frac{i}{n} \left(1 - \frac{1}{n} \right)^{n-1} \cdot 1.14. \quad (\text{A.106})$$

By applying these probabilities to Eq. (A.104), we have

$$\begin{aligned} &\mathbb{E}[x_t - x_{t+1} \mid x_t = i] \\ &= \left(1 - \frac{n-i+1}{n} \frac{i}{n} \right) P_{-1} + \sum_{d=2}^i dP_{-d} - \frac{i+1}{n} \frac{n-i}{n} P_1 - 2 \frac{i+2}{n} \frac{n-i}{n} P_2 \end{aligned}$$

$$\begin{aligned} &\leq \left(1 - \frac{n-i+1}{n} \frac{i}{n}\right) \frac{i}{n} \left(1 - \frac{1}{n}\right)^{n-1} \cdot 1.14 + \frac{i}{n} \left(\left(1 + \frac{1}{n}\right)^{i-1} - 1\right) \\ &\quad - \frac{i+1}{n} \frac{n-i}{n} \frac{n-i}{n} \left(1 - \frac{1}{n}\right)^{n-1} - 2 \frac{i+2}{n} \frac{n-i}{n} \frac{(n-i)(n-i-1)}{2n^2} \left(1 - \frac{1}{n}\right)^{n-2} \\ &\leq \frac{i}{n} \left(1 - \frac{1}{n}\right)^{n-1} \left(1.14 - 1 - 2 \cdot \frac{1}{2}\right) + O\left(\left(\frac{i}{n}\right)^2\right) \end{aligned} \quad (\text{A.107})$$

$$\leq -0.3 \cdot \frac{i}{n} + O\left(\left(\frac{i}{n}\right)^2\right), \quad (\text{A.108})$$

where Eq. (A.107) holds by $i < n^{1/4}$, and Eq. (A.108) holds by $(1 - 1/n)^{n-1} \geq 1/e$. To examine the condition of Theorem 2.6, we also need to analyze the probability $P(x_{t+1} \neq i | x_t = i)$ for $1 \leq i < n^{1/4}$. We have

$$P(x_{t+1} \neq i | x_t = i) \quad (\text{A.109})$$

$$= \left(1 - \frac{n-i+1}{n} \frac{i}{n}\right) P_{-1} + \sum_{d=2}^i P_{-d} + \frac{i+1}{n} \frac{n-i}{n} P_1 + \frac{i+2}{n} \frac{n-i}{n} P_2.$$

It can be verified that $P(x_{t+1} \neq i | x_t = i) = \Theta(i/n)$. Thus, we have $\mathbb{E}[x_t - x_{t+1} | x_t = i] = -\Omega(P(x_{t+1} \neq i | x_t = i))$, implying that condition (1) of Theorem 2.6 holds.

For condition (2) of Theorem 2.6, it requires to compare $P(|x_{t+1} - x_t| \geq j | x_t = i)$ with $(r(l)/(1 + \delta)^j) \cdot P(x_{t+1} \neq i | x_t = i)$ for $i \geq 1$. We rewrite $P(x_{t+1} \neq i | x_t = i)$ as $P(|x_{t+1} - x_t| \geq 1 | x_t = i)$, and show that condition (2) holds with $\delta = 1$ and $r(l) = 32e/7$. For $j \in \{1, 2, 3\}$, it holds because $r(l)/(1 + \delta)^j > 1$. For $j \geq 4$, according to the analysis on $p_{i,i+d}$, we have

$$\begin{aligned} P(|x_{t+1} - x_t| \geq j | x_t = i) &= \sum_{d=j}^i P_{-d} \\ &\leq \binom{i}{j} \frac{1}{n^j} \leq \frac{1}{j!} \left(\frac{i}{n}\right)^j \leq \frac{2}{2^j} \cdot \frac{i}{n}, \end{aligned} \quad (\text{A.110})$$

where Eq. (A.110) holds because for decreasing the number of 0-bits by at least j in mutation, it is necessary to flip at least j 0-bits. Furthermore,

$$\begin{aligned} P(|x_{t+1} - x_t| \geq 1 | x_t = i) &\geq p_{i,i-1} = \left(1 - \frac{n-i+1}{n} \frac{i}{n}\right) \cdot P_{-1} \\ &\geq \left(1 - \frac{n-i+1}{n} \frac{i}{n}\right) \cdot \frac{i}{n} \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{7}{16e} \cdot \frac{i}{n}. \end{aligned} \quad (\text{A.111})$$

Combining Eqs. (A.110) and (A.111), we have

$$\frac{r(l)}{(1 + \delta)^j} \cdot P(|x_{t+1} - x_t| \geq 1 | x_t = i)$$

$$\geq \frac{32e}{7} \frac{1}{2^j} \cdot \frac{7}{16e} \frac{i}{n} = \frac{2}{2^j} \frac{i}{n} \geq P(|x_{t+1} - x_t| \geq j \mid x_t = i), \quad (\text{A.112})$$

implying that condition (2) of Theorem 2.6 holds.

Note that $l = b - a = n^{1/4}$. Thus, by Theorem 2.6, the probability that the running time is $2^{O(n^{1/4})}$, is exponentially small when starting from a solution s with $|s|_0 \geq n^{1/4}$. Due to the uniform initial distribution, the probability that the initial solution s has $|s|_0 < n^{1/4}$ is exponentially small by the Chernoff bound. Thus, the expected running time is exponential. \square

Proof of Theorem 10.12.

We use Theorem 2.6 for the proof. We first analyze $p_{i,i+d}$ as in the proof of Theorem 10.11. Note that for a solution s , the fitness value output by sampling with $k = 2$ is $\hat{f}(s) = (f_1^n(s) + f_2^n(s))/2$, where $f_1^n(s)$ and $f_2^n(s)$ are noisy fitness values output by two independent fitness evaluations.

(1) When $d \geq 3$, $\hat{f}(s') \leq n - i - d + 1 \leq n - i - 2 < \hat{f}(s)$. Thus, the offspring solution s' will be discarded, and we have

$$\forall d \geq 3 : p_{i,i+d} = 0. \quad (\text{A.113})$$

(2) When $d = 2$, the offspring solution s' will be accepted iff $\hat{f}(s') = n - i - 1 = \hat{f}(s)$. The probability of $\hat{f}(s') = n - i - 1$ is $((i+2)/n)^2$, as it requires to flip one 0-bit of s' in both two noisy fitness evaluations. The probability of $\hat{f}(s) = n - i - 1$ is $((n-i)/n)^2$, as it requires to flip one 1-bit of s in both two noisy fitness evaluations. Thus,

$$p_{i,i+2} = P_2 \cdot \left(\frac{i+2}{n} \right)^2 \left(\frac{n-i}{n} \right)^2. \quad (\text{A.114})$$

(3) When $d = 1$, there are three possible cases for the acceptance of s' : $\hat{f}(s') = n - i \wedge \hat{f}(s) = n - i - 1$, $\hat{f}(s') = n - i \wedge \hat{f}(s) = n - i$ and $\hat{f}(s') = n - i - 1 \wedge \hat{f}(s) = n - i - 1$. The probability of $\hat{f}(s') = n - i$ is $((i+1)/n)^2$, as it requires to flip one 0-bit of s' in both two noisy evaluations. The probability of $\hat{f}(s') = n - i - 1$ is $2((i+1)/n)((n-i-1)/n)$, as it requires to flip one 0-bit of s' in one noisy evaluation and one 1-bit in the other noisy evaluation. Similarly, we can derive that the probabilities of $\hat{f}(s) = n - i - 1$ and $\hat{f}(s) = n - i$ are $((n-i)/n)^2$ and $2((n-i)/n)(i/n)$, respectively. Thus,

$$\begin{aligned} p_{i,i+1} = P_1 \cdot & \left(\left(\frac{i+1}{n} \right)^2 \left(\left(\frac{n-i}{n} \right)^2 + 2 \frac{n-i}{n} \frac{i}{n} \right) \right. \\ & \left. + 2 \frac{i+1}{n} \frac{n-i-1}{n} \left(\frac{n-i}{n} \right)^2 \right). \end{aligned} \quad (\text{A.115})$$

(4) When $d = -1$, s' will be rejected iff $\hat{f}(s') = n - i \wedge \hat{f}(s) = n - i + 1$. The probability of $\hat{f}(s') = n - i$ is $((n-i+1)/n)^2$, as it requires to flip one

1-bit of s' in both two noisy evaluations. The probability of $\hat{f}(s) = n - i + 1$ is $(i/n)^2$, as it requires to flip one 0-bit of s in both two noisy evaluations. Thus, we have

$$p_{i,i-1} = P_{-1} \cdot \left(1 - \left(\frac{n-i+1}{n} \right)^2 \left(\frac{i}{n} \right)^2 \right). \quad (\text{A.116})$$

(5) When $d \leq -2$, $\hat{f}(s') \geq n - i - d - 1 \geq n - i + 1 \geq \hat{f}(s)$. Thus, the offspring solution s' will always be accepted, and we have

$$\forall d \leq -2 : p_{i,i+d} = P_d. \quad (\text{A.117})$$

Using these probabilities, we have

$$\begin{aligned} \mathbb{E}[x_t - x_{t+1} \mid x_t = i] &= \sum_{d=1}^i d \cdot p_{i,i-d} - \sum_{d=1}^{n-i} d \cdot p_{i,i+d} \\ &= \left(1 - \left(\frac{n-i+1}{n} \right)^2 \left(\frac{i}{n} \right)^2 \right) P_{-1} + \sum_{d=2}^i d P_{-d} - 2 \left(\frac{i+2}{n} \right)^2 \left(\frac{n-i}{n} \right)^2 P_2 \\ &\quad - \left(\left(\frac{i+1}{n} \right)^2 \left(\left(\frac{n-i}{n} \right)^2 + 2 \frac{n-i}{n} \frac{i}{n} \right) + 2 \frac{i+1}{n} \frac{n-i-1}{n} \left(\frac{n-i}{n} \right)^2 \right) P_1 \\ &\leq \left(1 - \left(\frac{n-i+1}{n} \right)^2 \left(\frac{i}{n} \right)^2 \right) \frac{i}{n} \left(1 - \frac{1}{n} \right)^{n-1} \cdot 1.14 + \frac{i}{n} \left(\left(1 + \frac{1}{n} \right)^{i-1} - 1 \right) \\ &\quad - 2 \left(\frac{i+2}{n} \right)^2 \left(\frac{n-i}{n} \right)^2 \frac{(n-i)(n-i-1)}{2n^2} \left(1 - \frac{1}{n} \right)^{n-2} - \frac{n-i}{n} \left(1 - \frac{1}{n} \right)^{n-1} \\ &\quad \cdot \left(\left(\frac{i+1}{n} \right)^2 \left(\left(\frac{n-i}{n} \right)^2 + 2 \frac{n-i}{n} \frac{i}{n} \right) + 2 \frac{i+1}{n} \frac{n-i-1}{n} \left(\frac{n-i}{n} \right)^2 \right) \end{aligned} \quad (\text{A.118})$$

$$\leq \frac{i}{n} \left(1 - \frac{1}{n} \right)^{n-1} (1.14 - 2) + O \left(\left(\frac{i}{n} \right)^2 \right) \quad (\text{A.119})$$

$$\leq -0.3 \cdot \frac{i}{n} + O \left(\left(\frac{i}{n} \right)^2 \right), \quad (\text{A.120})$$

where Eq. (A.118) holds by using the bounds of P_d in the proof of Theorem 10.11, and Eq. (A.119) holds by $i < n^{1/4}$. It can be verified that $P(x_{t+1} \neq i \mid x_t = i) = \Theta(i/n)$ for $1 \leq i < n^{1/4}$. Thus, $\mathbb{E}[x_t - x_{t+1} \mid x_t = i] = -\Omega(P(x_{t+1} \neq i \mid x_t = i))$, implying that condition (1) of Theorem 2.6 holds.

Condition (2) of Theorem 2.6 holds with $\delta = 1$ and $r(l) = 32e/7$. The analytic procedure is the same as that in the proof of Theorem 10.11, because the following inequality holds:

$$\begin{aligned} P(|x_{t+1} - x_t| \geq 1 \mid x_t = i) &\geq p_{i,i-1} = \left(1 - \left(\frac{n-i+1}{n}\right)^2 \left(\frac{i}{n}\right)^2\right) \cdot P_{-1} \\ &\geq \left(1 - \frac{n-i+1}{n} \frac{i}{n}\right) \cdot P_{-1}. \end{aligned} \quad (\text{A.121})$$

Thus, by Theorem 2.6, the expected running time is exponential. \square

Proof of Theorem 10.15.

We use Theorem 2.3 for the proof. Let $\text{LO}(s) = \sum_{i=1}^n \prod_{j=1}^i s_j$ be the number of leading 1-bits of a solution $s \in \{0, 1\}^n$. Note that a state $x \in \mathcal{X}$ here is just a solution in $\{0, 1\}^n$. We construct a distance function as $\forall x \in \mathcal{X} = \{0, 1\}^n : V(x) = n - \text{LO}(x)$, satisfying that $V(x) = 0$ iff $x \in \mathcal{X}^* = \{1^n\}$.

We analyze $\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] \forall x : V(x) > 0$. For the current solution x , assume $\text{LO}(x) = i$, where $0 \leq i \leq n-1$. Let x' be the offspring solution generated by mutating x . Consider three mutation cases for $\text{LO}(x')$:

- (1) The l -th leading 1-bit is flipped and the first $(l-1)$ leading 1-bits remain unchanged, leading to $\text{LO}(x') = l-1$. Thus, we have $\forall 1 \leq l \leq i : P(\text{LO}(x') = l-1) = (1-1/n)^{l-1}(1/n)$.

- (2) The $(i+1)$ -th bit, which must be 0, is flipped and the first i leading 1-bits remain unchanged, leading to $\text{LO}(x') \geq i+1$. Thus, we have $P(\text{LO}(x') \geq i+1) = (1-1/n)^i(1/n)$.

- (3) The first $(i+1)$ bits remain unchanged, leading to $\text{LO}(x') = i$. Thus, we have $P(\text{LO}(x') = i) = (1-1/n)^{i+1}$.

Assume $\text{LO}(x') = j$. We then analyze the acceptance probability of x' , i.e., $P(\hat{f}(x') \geq \hat{f}(x))$. Note that $\hat{f}(x) = (\sum_{i=1}^k f_i^n(x))/k$, where $f_i^n(x)$ is the fitness output by one independent noisy evaluation. By one-bit noise with $p_n = 1/2$, the value of $f^n(x)$ can be calculated as follows:

- (1) Noise does not occur, with probability $1/2$. Thus, $P(f^n(x) = i) = 1/2$.

- (2) Noise occurs, with probability $p_n = 1/2$.

- (2.1) It flips the l -th leading 1-bit, leading to $f^n(x) = l-1$. Thus, we have $\forall 1 \leq l \leq i : P(f^n(x) = l-1) = 1/(2n)$.

- (2.2) It flips the $(i+1)$ -th bit, leading to $f^n(x) \geq i+1$. Thus, we have $P(f^n(x) \geq i+1) = 1/(2n)$. Note that $f^n(x)$ reaches the minimum $(i+1)$ when x has 0-bit at position $i+2$, and $f^n(x)$ reaches the maximum n when x has all 1-bits since position $i+2$.

- (2.3) Otherwise, $f^n(x)$ remains unchanged. Thus, we have $P(f^n(x) = i) = (1/2)(1 - (i+1)/n)$.

For each i , let x_i^{opt} be the solution which has only 1-bits except for the $(i+1)$ -th bit, i.e., $x_i^{\text{opt}} = 1^i 0 1^{n-i-1}$, and let x_i^{pes} be the solution with i leading 1-bits and otherwise only 0-bits, i.e., $x_i^{\text{pes}} = 1^i 0^{n-i}$. We have the stochastic ordering $f^n(x_i^{\text{pes}}) \preceq f^n(x) \preceq f^n(x_i^{\text{opt}})$, implying $\hat{f}(x_i^{\text{pes}}) \preceq \hat{f}(x) \preceq \hat{f}(x_i^{\text{opt}})$. Similarly, we have $\hat{f}(x_j^{\text{pes}}) \preceq \hat{f}(x') \preceq \hat{f}(x_j^{\text{opt}})$. Thus,

$$P(\hat{f}(x_j^{\text{pes}}) \geq \hat{f}(x_i^{\text{opt}})) \leq P(\hat{f}(x') \geq \hat{f}(x)) \leq P(\hat{f}(x_j^{\text{opt}}) \geq \hat{f}(x_i^{\text{pes}})). \quad (\text{A.122})$$

Let $P_{\text{mut}}(x, x')$ be the probability of generating x' by mutating x . By combining the mutation probability and the acceptance probability, we have

$$\begin{aligned} & \mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] \\ &= \sum_{j=0}^n \sum_{\text{LO}(x')=j} P_{\text{mut}}(x, x') \cdot P(\hat{f}(x') \geq \hat{f}(x)) \cdot (n - i - (n - j)) \\ &\geq \sum_{j=0}^{i-1} \sum_{\text{LO}(x')=j} P_{\text{mut}}(x, x') \cdot P(\hat{f}(x_j^{\text{opt}}) \geq \hat{f}(x_i^{\text{pes}})) \cdot (j - i) \\ &\quad + \sum_{j=i+1}^n \sum_{\text{LO}(x')=j} P_{\text{mut}}(x, x') \cdot P(\hat{f}(x_j^{\text{pes}}) \geq \hat{f}(x_i^{\text{opt}})) \cdot (j - i) \end{aligned} \quad (\text{A.123})$$

$$\begin{aligned} &\geq \sum_{j=0}^{i-1} \sum_{\text{LO}(x')=j} P_{\text{mut}}(x, x') \cdot P(\hat{f}(x_{i-1}^{\text{opt}}) \geq \hat{f}(x_i^{\text{pes}})) \cdot (j - i) \\ &\quad + \sum_{j=i+1}^n \sum_{\text{LO}(x')=j} P_{\text{mut}}(x, x') \cdot P(\hat{f}(x_{i+1}^{\text{pes}}) \geq \hat{f}(x_i^{\text{opt}})) \cdot (i + 1 - i) \end{aligned} \quad (\text{A.124})$$

$$\begin{aligned} &= \left(1 - \frac{1}{n}\right)^i \frac{1}{n} P(\hat{f}(x_{i+1}^{\text{pes}}) \geq \hat{f}(x_i^{\text{opt}})) \\ &\quad - \left(\sum_{j=0}^{i-1} \left(1 - \frac{1}{n}\right)^j \frac{1}{n} (i - j) \right) P(\hat{f}(x_{i-1}^{\text{opt}}) \geq \hat{f}(x_i^{\text{pes}})) \end{aligned} \quad (\text{A.125})$$

$$\geq \frac{1}{en} P(\hat{f}(x_{i+1}^{\text{pes}}) \geq \hat{f}(x_i^{\text{opt}})) - \frac{i(i+1)}{2n} P(\hat{f}(x_{i-1}^{\text{opt}}) \geq \hat{f}(x_i^{\text{pes}})). \quad (\text{A.126})$$

Note that Eq. (A.123) holds by Eq. (A.122); Eq. (A.124) holds because $\forall j \leq i-1 : P(\hat{f}(x_j^{\text{opt}}) \geq \hat{f}(x_i^{\text{pes}})) \leq P(\hat{f}(x_{i-1}^{\text{opt}}) \geq \hat{f}(x_i^{\text{pes}}))$ and $\forall j \geq i+1 : P(\hat{f}(x_j^{\text{pes}}) \geq \hat{f}(x_i^{\text{opt}})) \geq P(\hat{f}(x_{i+1}^{\text{pes}}) \geq \hat{f}(x_i^{\text{opt}}))$; Eq. (A.125) holds because $\sum_{j=i+1}^n \sum_{\text{LO}(x')=j} P_{\text{mut}}(x, x') = P(\text{LO}(x') \geq i+1) = (1 - 1/n)^i (1/n)$ and $\forall 0 \leq j \leq i-1 : \sum_{\text{LO}(x')=j} P_{\text{mut}}(x, x') = P(\text{LO}(x') = j) = (1 - 1/n)^j (1/n)$; Eq. (A.126) holds by $i \leq n-1$ and $1 \geq (1 - 1/n)^i \geq (1 - 1/n)^{n-1} \geq 1/e$.

Next we bound the probabilities $P(\hat{f}(x_{i+1}^{\text{pes}}) \geq \hat{f}(x_i^{\text{opt}}))$ and $P(\hat{f}(x_{i-1}^{\text{opt}}) \geq \hat{f}(x_i^{\text{pes}}))$. First, we have

$$\begin{aligned} P(\hat{f}(x') \geq \hat{f}(x)) &= P\left(\left(\sum_{i=1}^k f_i^n(x')\right)/k \geq \left(\sum_{i=1}^k f_i^n(x)\right)/k\right) \\ &= P\left(\sum_{i=1}^k f_i^n(x') - \sum_{i=1}^k f_i^n(x) \geq 0\right) = P(Z(x', x) \geq 0), \end{aligned} \quad (\text{A.127})$$

where the random variable $Z(x', x)$ is used to represent $(\sum_{i=1}^k f_i^n(x') - \sum_{i=1}^k f_i^n(x))$ for notational convenience. We then calculate the expectation

and variance of $f^n(x_i^{\text{opt}})$ and $f^n(x_i^{\text{pes}})$, respectively. Based on the analysis of $f^n(x)$, we have

$$\begin{aligned}\mathbb{E}[f^n(x_i^{\text{opt}})] &= \frac{1}{2}i + \sum_{j=0}^{i-1} \frac{1}{2n}j + \frac{1}{2n}n + \frac{1}{2} \left(1 - \frac{i+1}{n}\right) i \\ &= i + \frac{1}{2} - \frac{i^2 + 3i}{4n},\end{aligned}\tag{A.128}$$

$$\begin{aligned}\mathbb{E}[f^n(x_i^{\text{pes}})] &= \frac{1}{2}i + \sum_{j=0}^{i-1} \frac{1}{2n}j + \frac{1}{2n}(i+1) + \frac{1}{2} \left(1 - \frac{i+1}{n}\right) i \\ &= i - \frac{i^2 + i - 2}{4n},\end{aligned}\tag{A.129}$$

$$\begin{aligned}\text{Var}(f^n(x_i^{\text{opt}})) &= \mathbb{E}[(f^n(x_i^{\text{opt}}))^2] - (\mathbb{E}[f^n(x_i^{\text{opt}})])^2 \\ &= \frac{1}{2}i^2 + \sum_{j=0}^{i-1} \frac{1}{2n}j^2 + \frac{1}{2n}n^2 + \frac{1}{2} \left(1 - \frac{i+1}{n}\right) i^2 - \left(i + \frac{1}{2} - \frac{i^2 + 3i}{4n}\right)^2 \\ &\leq \frac{1}{6}n^2 + \frac{3}{2}n + \frac{5}{6} \leq \frac{7}{6}n^2,\end{aligned}\tag{A.130}$$

$$\begin{aligned}\text{Var}(f^n(x_i^{\text{pes}})) &= \mathbb{E}[(f^n(x_i^{\text{pes}}))^2] - (\mathbb{E}[f^n(x_i^{\text{pes}})])^2 \\ &= \frac{1}{2}i^2 + \sum_{j=0}^{i-1} \frac{1}{2n}j^2 + \frac{1}{2n}(i+1)^2 + \frac{1}{2} \left(1 - \frac{i+1}{n}\right) i^2 - \left(i - \frac{i^2 + i - 2}{4n}\right)^2 \\ &\leq \frac{1}{6}n^2 + \frac{1}{4}n + \frac{1}{12} + \frac{1}{2n} \leq \frac{5}{12}n^2.\end{aligned}\tag{A.131}$$

Note that the last inequalities of Eqs. (A.130) and (A.131) hold with $n \geq 2$. Thus, we have

$$\mathbb{E}[Z(x_{i+1}^{\text{pes}}, x_i^{\text{opt}})] = k(\mathbb{E}[f^n(x_{i+1}^{\text{pes}})] - \mathbb{E}[f^n(x_i^{\text{opt}})]) = \frac{k}{2},\tag{A.132}$$

$$\mathbb{E}[Z(x_{i-1}^{\text{opt}}, x_i^{\text{pes}})] = k(\mathbb{E}[f^n(x_{i-1}^{\text{opt}})] - \mathbb{E}[f^n(x_i^{\text{pes}})]) = -\frac{k}{2},\tag{A.133}$$

$$\text{Var}(Z(x_{i+1}^{\text{pes}}, x_i^{\text{opt}})) = k(\text{Var}(f^n(x_{i+1}^{\text{pes}})) + \text{Var}(f^n(x_i^{\text{opt}}))) \leq \frac{19}{12}kn^2,\tag{A.134}$$

$$\text{Var}(Z(x_{i-1}^{\text{opt}}, x_i^{\text{pes}})) = k(\text{Var}(f^n(x_{i-1}^{\text{opt}})) + \text{Var}(f^n(x_i^{\text{pes}}))) \leq \frac{19}{12}kn^2.\tag{A.135}$$

Now, we can derive the bounds of the probabilities $P(\hat{f}(x_{i+1}^{\text{pes}}) \geq \hat{f}(x_i^{\text{opt}}))$ and $P(\hat{f}(x_{i-1}^{\text{opt}}) \geq \hat{f}(x_i^{\text{pes}}))$ by Chebyshev's inequality. Note that $Z(x', x)$ is integer-valued. We have

$$\begin{aligned}P(\hat{f}(x_{i+1}^{\text{pes}}) \geq \hat{f}(x_i^{\text{opt}})) &= P(Z(x_{i+1}^{\text{pes}}, x_i^{\text{opt}}) \geq 0) = 1 - P(Z(x_{i+1}^{\text{pes}}, x_i^{\text{opt}}) \leq -1) \\ &= 1 - P(Z(x_{i+1}^{\text{pes}}, x_i^{\text{opt}}) - \mathbb{E}[Z(x_{i+1}^{\text{pes}}, x_i^{\text{opt}})] \leq -1 - \mathbb{E}[Z(x_{i+1}^{\text{pes}}, x_i^{\text{opt}})]) \\ &\geq 1 - P(|Z(x_{i+1}^{\text{pes}}, x_i^{\text{opt}}) - \mathbb{E}[Z(x_{i+1}^{\text{pes}}, x_i^{\text{opt}})]| \geq 1 + k/2)\end{aligned}$$

$$\geq 1 - \frac{\text{Var}(Z(x_{i+1}^{\text{pes}}, x_i^{\text{opt}}))}{(1 + k/2)^2} \geq 1 - \frac{19n^2}{3k}, \quad (\text{A.136})$$

where Eq. (A.136) is derived by applying Chebyshev's inequality. Similarly, we have

$$\begin{aligned} P(\hat{f}(x_{i-1}^{\text{opt}}) \geq \hat{f}(x_i^{\text{pes}})) &= P(Z(x_{i-1}^{\text{opt}}, x_i^{\text{pes}}) \geq 0) \\ &= P(Z(x_{i-1}^{\text{opt}}, x_i^{\text{pes}}) - \mathbb{E}[Z(x_{i-1}^{\text{opt}}, x_i^{\text{pes}})] \geq -\mathbb{E}[Z(x_{i-1}^{\text{opt}}, x_i^{\text{pes}})]) \\ &\leq P(|Z(x_{i-1}^{\text{opt}}, x_i^{\text{pes}}) - \mathbb{E}[Z(x_{i-1}^{\text{opt}}, x_i^{\text{pes}})]| \geq k/2) \\ &\leq \frac{\text{Var}(Z(x_{i-1}^{\text{opt}}, x_i^{\text{pes}}))}{(k/2)^2} \leq \frac{19n^2}{3k}, \end{aligned} \quad (\text{A.137})$$

where Eq. (A.137) is derived by applying Chebyshev's inequality.

By applying Eqs. (A.136) and (A.137) to Eq. (A.126), we have

$$\begin{aligned} \mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] &\geq \frac{1}{en} \left(1 - \frac{19n^2}{3k} \right) - \frac{i(i+1)}{2n} \frac{19n^2}{3k} \\ &\geq \frac{1}{en} \left(1 - \frac{19}{30n^2} \right) - \frac{(n-1)n}{2n} \frac{19}{30n^2} \end{aligned} \quad (\text{A.138})$$

$$\geq \frac{0.05}{n} - \frac{19}{30en^3}, \quad (\text{A.139})$$

where Eq. (A.138) holds by $i \leq n-1$ and $k = 10n^4$. Thus, the condition of Theorem 2.3 holds with $\mathbb{E}[V(\xi_t) - V(\xi_{t+1}) \mid \xi_t = x] = \Omega(1/n)$. Considering $V(x) = n - \text{LO}(x) \leq n$, we have

$$\mathbb{E}[\tau \mid \xi_0] = O(n) \cdot V(\xi_0) = O(n^2), \quad (\text{A.140})$$

i.e., the expected number of iterations for (1+1)-EA finding the optimal solution is $O(n^2)$. Because the expected running time is $2k$, i.e., the number of fitness evaluations in each iteration, times the expected number of iterations and $k = 10n^4$, the expected running time is $O(n^6)$. \square

Proof of Theorem 10.16.

We use Theorem 2.5 for the proof. Let x_t be the number of 0-bits of the solution after t iterations of (1+1)-EA. We consider the interval $[0, n^{1/4}]$, i.e., the parameters $a = 0$ and $b = n^{1/4}$ in Theorem 2.5, and analyze the drift $\mathbb{E}[x_t - x_{t+1} \mid x_t = i]$ for $1 \leq i < n^{1/4}$. Let $p_{i,i+d}$ denote the probability for the next solution after bit-wise mutation and selection to have $(i+d)$ number of 0-bits, i.e., $x_{t+1} = i + d$, where $-i \leq d \leq n - i$; let P_d denote the probability for the offspring solution generated by bit-wise mutation to have $(i+d)$ number of 0-bits, i.e., $|s'|_0 = i + d$. $\forall d \neq 0$, we have

$$\begin{aligned} p_{i,i+d} &= P_d \cdot P(f^n(s') \geq f^n(s)) = P_d \cdot P(n - i - d + \delta_1 \geq n - i + \delta_2) \\ &= P_d \cdot P(\delta_1 - \delta_2 \geq d) = P_d \cdot P(\delta \geq d), \end{aligned} \quad (\text{A.141})$$

where $\delta_1, \delta_2 \sim \mathcal{N}(\theta, \sigma^2)$ and $\delta \sim \mathcal{N}(0, 2\sigma^2)$. Thus, we have

$$\begin{aligned}\mathbb{E}[x_t - x_{t+1} \mid x_t = i] &= \sum_{d=1}^i d \cdot p_{i,i-d} - \sum_{d=1}^{n-i} d \cdot p_{i,i+d} \\ &\leq \sum_{d=1}^i d \cdot p_{i,i-d} - p_{i,i+1} \leq \sum_{d=1}^i d \cdot P_{-d} - P_1 \cdot \mathbb{P}(\delta \geq 1).\end{aligned}\tag{A.142}$$

Let $\delta' \sim \mathcal{N}(0, 1)$. It holds that $\mathbb{P}(\delta \geq 1) = \mathbb{P}(\delta' \geq 1/(\sqrt{2}\sigma)) \geq \mathbb{P}(\delta' \geq 1/\sqrt{2}) = \mathbb{P}(\delta' \leq -1/\sqrt{2}) \geq 0.23$, where the first inequality holds by $\sigma \geq 1$, and the last inequality is obtained by calculating the cumulative distribution function of the standard normal distribution. Furthermore, $P_1 \geq ((n-i)/n)(1-1/n)^{n-1} \geq (n-i)/(en)$, and $P_{-d} \leq \binom{i}{d}(1/n^d)$. Applying these probability bounds to Eq. (A.142), we have

$$\begin{aligned}\mathbb{E}[x_t - x_{t+1} \mid x_t = i] &\leq \sum_{d=1}^i d \cdot \binom{i}{d} \frac{1}{n^d} - \frac{n-i}{en} \cdot 0.23 \\ &= -\frac{0.23}{e} + \frac{0.23}{e} \frac{i}{n} + \frac{i}{n} \left(1 + \frac{1}{n}\right)^{i-1} \leq -\frac{0.23}{e} + \left(\frac{0.23}{e} + e\right) \frac{i}{n} \\ &= -\frac{0.23}{e} + O\left(\frac{n^{1/4}}{n}\right),\end{aligned}\tag{A.143}$$

where Eq. (A.143) holds by $i < n^{1/4}$. In other words, $\mathbb{E}[x_t - x_{t+1} \mid x_t = i] = -\Omega(1)$, implying that condition (1) of Theorem 2.5 holds. As the analysis of Eq. (A.98) in the proof of Theorem 10.9, condition (2) of Theorem 2.5 holds with $\delta = 1$ and $r(l) = 2$. Note that $l = b - a = n^{1/4}$. Thus, by Theorem 2.5, the expected running time is exponential. \square

Proof of Theorem 10.17.

We use Theorem 2.5 for the proof. Let x_t be the number of 0-bits of the solution after t iterations of (1+1)-EA. As in the proof of Theorem 10.16,

$$\mathbb{E}[x_t - x_{t+1} \mid x_t = i] \leq \sum_{d=1}^i d \cdot p_{i,i-d} - p_{i,i+1}.\tag{A.144}$$

It holds that $p_{i,i-d} \leq P_{-d} \leq \binom{i}{d}(1/n^d)$. For $p_{i,i+1}$, consider $(n-i)$ cases where only one 1-bit is flipped, occurring with probability $(1/n)(1-1/n)^{n-1}$. Let s and s' denote the current solution and the offspring solution, respectively. Let LO denote the number of leading 1-bits of s , i.e., $LO = f(s)$. The noisy fitness of s is $f^n(s) = LO + \delta_1$, where $\delta_1 \sim \mathcal{N}(\theta, \sigma^2)$. The acceptance probability of s' in these $(n-i)$ cases can be calculated as follows:

- (1) If the flipped 1-bit is the j -th leading 1-bit, $f^n(s') = j - 1 + \delta_2$, where $1 \leq j \leq LO$ and $\delta_2 \sim \mathcal{N}(\theta, \sigma^2)$. Thus, the acceptance probability is

$P(f^n(s') \geq f^n(s)) = P(j - 1 + \delta_2 \geq LO + \delta_1) = P(\delta_2 - \delta_1 \geq LO - j + 1) = P(\delta \geq LO - j + 1)$, where $\delta \sim \mathcal{N}(0, 2\sigma^2)$.

(2) Otherwise, $f^n(s') = LO + \delta_2$, and the acceptance probability is $P(\delta \geq 0)$.

Applying these probability bounds to Eq. (A.144), we have

$$\begin{aligned} \mathbb{E}[x_t - x_{t+1} \mid x_t = i] &\leq \sum_{d=1}^i d \cdot \binom{i}{d} \frac{1}{n^d} \\ &\quad - \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} \cdot \left(\sum_{j=1}^{LO} P(\delta \geq LO - j + 1) + (n - i - LO)P(\delta \geq 0) \right) \\ &\leq \frac{i}{n} \left(1 + \frac{1}{n}\right)^{i-1} - \frac{1}{en} \cdot \sum_{j=1}^{n-i} P(\delta \geq n - i - j + 1), \end{aligned} \quad (\text{A.145})$$

where Eq. (A.145) holds because the term in big brackets reaches the minimum when $LO = n - i$. Let $\delta' \sim \mathcal{N}(0, 1)$. By Eq. (D.17) in [Mohri et al., 2012], $P(\delta' \geq u) \geq (1/2)(1 - \sqrt{1 - e^{-u^2}}) \geq (1/2)(1 - u)$ for $u > 0$, and thus,

$$\begin{aligned} P(\delta \geq n - i - j + 1) &= P\left(\frac{\delta}{\sqrt{2}\sigma} \geq \frac{n - i - j + 1}{\sqrt{2}\sigma}\right) = P\left(\delta' \geq \frac{n - i - j + 1}{\sqrt{2}\sigma}\right) \\ &\geq \frac{1}{2} \left(1 - \frac{n - i - j + 1}{\sqrt{2}\sigma}\right) \geq \frac{1}{2} \left(1 - \frac{n - i - j + 1}{\sqrt{2}n}\right), \end{aligned} \quad (\text{A.146})$$

where Eq. (A.146) holds by $\sigma \geq n$. Thus, we have

$$\begin{aligned} \mathbb{E}[x_t - x_{t+1} \mid x_t = i] &\leq \frac{i}{n} \left(1 + \frac{1}{n}\right)^{i-1} - \frac{1}{en} \cdot \sum_{j=1}^{n-i} \frac{1}{2} \left(1 - \frac{n - i - j + 1}{\sqrt{2}n}\right) \\ &= \frac{i}{n} \left(1 + \frac{1}{n}\right)^{i-1} - \frac{n - i}{2en} \cdot \left(1 - \frac{n - i + 1}{2\sqrt{2}n}\right) \\ &\leq -\frac{1}{2e} \left(1 - \frac{1}{2\sqrt{2}}\right) + O\left(\frac{n^{1/4}}{n}\right), \end{aligned} \quad (\text{A.147})$$

where Eq. (A.147) holds by $i < n^{1/4}$. In other words, $\mathbb{E}[x_t - x_{t+1} \mid x_t = i] = -\Omega(1)$, implying that condition (1) of Theorem 2.5 holds. As the analysis of Eq. (A.98) in the proof of Theorem 10.9, it can be verified that condition (2) of Theorem 2.5 holds with $\delta = 1$ and $r(l) = 2$. Thus, the expected running time is exponential. \square

Proof of Corollary 10.2.

As in the proof of Corollary 10.1, sampling reduces the variance σ^2 of noise to σ^2/k . As $k = \lceil 12en^2\sigma^2 \rceil$, $\sigma^2/k \leq 1/(12en^2)$. By Lemma 10.7, the expected number of iterations of (1+1)-EA for finding the optimal solution is $O(n^2)$. The expected running time is $2k$ times the expected number of iterations, and thus, polynomial, due to $\sigma^2 = O(\text{poly}(n))$. \square

A.6 Proofs in Chapter 11

Proof of Lemma 11.1.

Let $f(m) = \sum_{k=0}^i \binom{m}{k} (1/n)^k (1 - 1/n)^{m-k}$. The goal is to show that $f(m+1) \leq f(m)$ for $m \geq i$. Denote x_1, x_2, \dots, x_{m+1} as independent random variables, where x_j satisfies that $P(x_j = 1) = 1/n$ and $P(x_j = 0) = 1 - 1/n$. Because $f(m)$ and $f(m+1)$ can be expressed as $f(m) = P(\sum_{j=1}^m x_j \leq i)$ and $f(m+1) = P(\sum_{j=1}^{m+1} x_j \leq i)$, we have

$$\begin{aligned} f(m+1) &= P\left(\sum_{j=1}^m x_j < i\right) + P\left(\sum_{j=1}^m x_j = i\right) P(x_{m+1} = 0) \\ &= P\left(\sum_{j=1}^m x_j < i\right) + P\left(\sum_{j=1}^m x_j = i\right) \left(1 - \frac{1}{n}\right) \leq f(m). \end{aligned} \quad (\text{A.148})$$

□

Proof of Lemma 11.2.

Let $m = \lceil e(c+1) \log n / \log \log n \rceil$. Denote x_1, x_2, \dots, x_n as independent random variables, where $P(x_j = 1) = 1/n$ and $P(x_j = 0) = 1 - 1/n$. Let $x = \sum_{j=1}^n x_j$. The expectation $\mathbb{E}[x] = 1$, and

$$\forall i \geq m : \sum_{k=i}^n \binom{n}{k} \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} = P(x \geq i) \leq \frac{e^{(i-1)}}{i^i}, \quad (\text{A.149})$$

where the inequality holds by the Chernoff bound. Then, we have

$$\begin{aligned} &\sum_{i=0}^{n-1} \left(\sum_{k=0}^i \binom{n}{k} \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \right)^\lambda \\ &\geq \sum_{i=m-1}^{n-1} \left(\sum_{k=0}^i \binom{n}{k} \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \right)^\lambda \\ &= \sum_{i=m-1}^{n-1} \left(1 - \sum_{k=i+1}^n \binom{n}{k} \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \right)^\lambda \\ &\geq \sum_{i=m-1}^{n-1} \left(1 - \frac{e^i}{(i+1)^{(i+1)}} \right)^\lambda \geq \sum_{i=m-1}^{n-1} \left(1 - \frac{e^{m-1}}{m^m} \right)^\lambda, \end{aligned} \quad (\text{A.150})$$

where Eq. (A.150) holds by Eq. (A.149) and $e^i/(i+1)^{(i+1)}$ decreasing with i when $i \geq m-1$. We evaluate e^{m-1}/m^m by taking logarithm to its reciprocal.

$$\log(m^m/e^{m-1}) = m(\log m - 1) + 1$$

$$\begin{aligned} &\geq \frac{e(c+1)\log n}{\log \log n}(1 + \log(c+1) + \log \log n - \log \log \log n - 1) + 1 \\ &\geq \frac{e(c+1)\log n}{\log \log n} \frac{1}{e} \log \log n = (c+1)\log n \geq \log(\lambda n), \end{aligned} \quad (\text{A.151})$$

where Eq. (A.151) holds by $\lambda \leq n^c$. Thus, $e^{m-1}/m^m \leq 1/(\lambda n)$, implying

$$\begin{aligned} \sum_{i=0}^{n-1} \left(\sum_{k=0}^i \binom{n}{k} \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \right)^\lambda &\geq \sum_{i=m-1}^{n-1} \left(1 - \frac{1}{\lambda n}\right)^\lambda \\ &\geq \sum_{i=m-1}^{n-1} \left(1 - \frac{1}{n}\right) \geq n - m \end{aligned} \quad (\text{A.152})$$

$$= n - \left\lceil \frac{e(c+1)\log n}{\log \log n} \right\rceil, \quad (\text{A.153})$$

where Eq. (A.152) holds by $\forall 0 \leq xy \leq 1, y \geq 1 : (1-x)^y \geq 1 - xy$. \square

A.7 Proofs in Chapter 12

Proof of Theorem 12.3.

First, consider the one-step expectation. We have

$$\begin{aligned} \forall i : \mathbb{E}[\mathcal{G}(\xi_{t+1}) \mid \xi_t \in \mathcal{X}_i] - \mathbb{E}[\mathcal{G}(\xi_{t+1}^F) \mid \xi_t^F \in \mathcal{X}_i^F] \\ = \sum_{x \in \mathcal{X}_i} \mathbb{E}[\mathcal{G}(\xi_{t+1}) \mid \xi_t = x] P(\xi_t = x \mid \xi_t \in \mathcal{X}_i) \\ - \sum_{x \in \mathcal{X}_i^F} \mathbb{E}[\mathcal{G}(\xi_{t+1}^F) \mid \xi_t^F = x] P(\xi_t^F = x \mid \xi_t^F \in \mathcal{X}_i^F) \\ = \sum_{x \in \mathcal{X}_i} \sum_{j=1}^m \mathcal{G}(\mathcal{X}_j) P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) P(\xi_t = x \mid \xi_t \in \mathcal{X}_i) \\ - \sum_{x \in \mathcal{X}_i^F} \sum_{j=1}^m \mathcal{G}(\mathcal{X}_j) P(\xi_{t+1}^F \in \mathcal{X}_j \mid \xi_t^F = x) P(\xi_t^F = x \mid \xi_t^F \in \mathcal{X}_i^F) \\ \geq 0, \end{aligned} \quad (\text{A.154})$$

where Eq. (A.154) holds by Eq. (A.155). $\forall x \in \mathcal{X}_i, y \in \mathcal{X}_i^F$, by Eq. (12.23) and $\mathcal{G}(\mathcal{X}_j)$ increasing with j , we have $\exists q \in [0, 1], h \in [m]$ such that

$$\begin{aligned} \sum_{j=1}^m \mathcal{G}(\mathcal{X}_j) P(\xi_{t+1} \in \mathcal{X}_j \mid \xi_t = x) - \sum_{j=1}^m \mathcal{G}(\mathcal{X}_j) P(\xi_{t+1}^F \in \mathcal{X}_j \mid \xi_t^F = y) \\ \geq (\mathcal{G}(\mathcal{X}_h) - \mathcal{G}(\mathcal{X}_{h-1}))q \geq 0. \end{aligned} \quad (\text{A.155})$$

Similarly, we have

$$\begin{aligned}
& \forall i > j : \mathbb{E}[\mathcal{G}(\xi_{t+1}) \mid \xi_t \in \mathcal{X}_i] - \mathbb{E}[\mathcal{G}(\xi_{t+1}) \mid \xi_t \in \mathcal{X}_j] \\
&= \sum_{x \in \mathcal{X}_i} \mathbb{E}[\mathcal{G}(\xi_{t+1}) \mid \xi_t = x] P(\xi_t = x \mid \xi_t \in \mathcal{X}_i) \\
&\quad - \sum_{x \in \mathcal{X}_j} \mathbb{E}[\mathcal{G}(\xi_{t+1}) \mid \xi_t = x] P(\xi_t = x \mid \xi_t \in \mathcal{X}_j) \\
&= \sum_{x \in \mathcal{X}_i} \sum_{k=1}^m \mathcal{G}(\mathcal{X}_k) P(\xi_{t+1} \in \mathcal{X}_k \mid \xi_t = x) P(\xi_t = x \mid \xi_t \in \mathcal{X}_i) \\
&\quad - \sum_{x \in \mathcal{X}_j} \sum_{k=1}^m \mathcal{G}(\mathcal{X}_k) P(\xi_{t+1} \in \mathcal{X}_k \mid \xi_t = x) P(\xi_t = x \mid \xi_t \in \mathcal{X}_j) \\
&\geq 0,
\end{aligned} \tag{A.156}$$

where Eq. (A.156) holds by Eq. (A.157). $\forall x \in \mathcal{X}_i, y \in \mathcal{X}_j$ where $i > j$, by Eq. (12.22) and $\mathcal{G}(\mathcal{X}_j)$ increasing with j , we have $\exists q \in [0, 1], h \in [m]$:

$$\begin{aligned}
& \sum_{k=1}^m \mathcal{G}(\mathcal{X}_k) P(\xi_{t+1} \in \mathcal{X}_k \mid \xi_t = x) - \sum_{k=1}^m \mathcal{G}(\mathcal{X}_k) P(\xi_{t+1} \in \mathcal{X}_k \mid \xi_t = y) \\
&\geq (\mathcal{G}(\mathcal{X}_h) - \mathcal{G}(\mathcal{X}_{h-1}))q \geq 0.
\end{aligned} \tag{A.157}$$

Thus, we have

$$\forall i : \mathbb{E}[\mathcal{G}(\xi_{t+1}) \mid \xi_t \in \mathcal{X}_i] \geq \mathbb{E}[\mathcal{G}(\xi_{t+1}^F) \mid \xi_t^F \in \mathcal{X}_i^F], \tag{A.158}$$

$$\forall i > j : \mathbb{E}[\mathcal{G}(\xi_{t+1}) \mid \xi_t \in \mathcal{X}_i] \geq \mathbb{E}[\mathcal{G}(\xi_{t+1}) \mid \xi_t \in \mathcal{X}_j]. \tag{A.159}$$

Inductively assume that

$$\forall i : \mathbb{E}[\mathcal{G}(\xi_{t+h}) \mid \xi_t \in \mathcal{X}_i] \geq \mathbb{E}[\mathcal{G}(\xi_{t+h}^F) \mid \xi_t^F \in \mathcal{X}_i^F], \tag{A.160}$$

$$\forall i > j : \mathbb{E}[\mathcal{G}(\xi_{t+h}) \mid \xi_t \in \mathcal{X}_i] \geq \mathbb{E}[\mathcal{G}(\xi_{t+h}) \mid \xi_t \in \mathcal{X}_j]. \tag{A.161}$$

Because it holds that

$$\begin{aligned}
& \mathbb{E}[\mathcal{G}(\xi_{t+h+1}) \mid \xi_t \in \mathcal{X}_i] = \sum_{j=1}^m \mathcal{G}(\mathcal{X}_j) P(\xi_{t+h+1} \in \mathcal{X}_j \mid \xi_t \in \mathcal{X}_i) \\
&= \sum_{j=1}^m \mathcal{G}(\mathcal{X}_j) \sum_{z=1}^m P(\xi_{t+h+1} \in \mathcal{X}_j \mid \xi_{t+1} \in \mathcal{X}_z) P(\xi_{t+1} \in \mathcal{X}_z \mid \xi_t \in \mathcal{X}_i) \\
&= \sum_{z=1}^m P(\xi_{t+1} \in \mathcal{X}_z \mid \xi_t \in \mathcal{X}_i) \sum_{j=1}^m \mathcal{G}(\mathcal{X}_j) P(\xi_{t+h+1} \in \mathcal{X}_j \mid \xi_{t+1} \in \mathcal{X}_z) \\
&= \sum_{z=1}^m P(\xi_{t+1} \in \mathcal{X}_z \mid \xi_t \in \mathcal{X}_i) \mathbb{E}[\mathcal{G}(\xi_{t+h}) \mid \xi_t \in \mathcal{X}_z],
\end{aligned} \tag{A.162}$$

we have

$$\begin{aligned}
& \forall i : \mathbb{E}[\mathcal{G}(\xi_{t+h+1}) \mid \xi_t \in \mathcal{X}_i] - \mathbb{E}[\mathcal{G}(\xi_{t+h+1}^F) \mid \xi_t^F \in \mathcal{X}_i^F] \\
&= \sum_{z=1}^m P(\xi_{t+1} \in \mathcal{X}_z \mid \xi_t \in \mathcal{X}_i) \mathbb{E}[\mathcal{G}(\xi_{t+h}) \mid \xi_t \in \mathcal{X}_z] \\
&\quad - \sum_{z=1}^m P(\xi_{t+1}^F \in \mathcal{X}_z^F \mid \xi_t^F \in \mathcal{X}_i^F) \mathbb{E}[\mathcal{G}(\xi_{t+h}^F) \mid \xi_t^F \in \mathcal{X}_z^F] \\
&\geq \sum_{z=1}^m P(\xi_{t+1} \in \mathcal{X}_z \mid \xi_t \in \mathcal{X}_i) \mathbb{E}[\mathcal{G}(\xi_{t+h}) \mid \xi_t \in \mathcal{X}_z] \\
&\quad - \sum_{z=1}^m P(\xi_{t+1}^F \in \mathcal{X}_z^F \mid \xi_t^F \in \mathcal{X}_i^F) \mathbb{E}[\mathcal{G}(\xi_{t+h}^F) \mid \xi_t^F \in \mathcal{X}_z^F] \quad (\text{A.163}) \\
&\geq 0, \quad (\text{A.164})
\end{aligned}$$

where Eq. (A.163) holds by inductive hypothesis $\forall i : \mathbb{E}[\mathcal{G}(\xi_{t+h}) \mid \xi_t \in \mathcal{X}_i] \geq \mathbb{E}[\mathcal{G}(\xi_{t+h}^F) \mid \xi_t^F \in \mathcal{X}_i^F]$, and Eq. (A.164) holds by Eq. (12.23) and inductive hypothesis $\forall i > j : \mathbb{E}[\mathcal{G}(\xi_{t+h}) \mid \xi_t \in \mathcal{X}_i] \geq \mathbb{E}[\mathcal{G}(\xi_{t+h}) \mid \xi_t \in \mathcal{X}_j]$. Similarly,

$$\begin{aligned}
& \forall i > j : \mathbb{E}[\mathcal{G}(\xi_{t+h+1}) \mid \xi_t \in \mathcal{X}_i] - \mathbb{E}[\mathcal{G}(\xi_{t+h+1}) \mid \xi_t \in \mathcal{X}_j] \\
&= \sum_{z=1}^m P(\xi_{t+1} \in \mathcal{X}_z \mid \xi_t \in \mathcal{X}_i) \mathbb{E}[\mathcal{G}(\xi_{t+h}) \mid \xi_t \in \mathcal{X}_z] \\
&\quad - \sum_{z=1}^m P(\xi_{t+1} \in \mathcal{X}_z \mid \xi_t \in \mathcal{X}_j) \mathbb{E}[\mathcal{G}(\xi_{t+h}) \mid \xi_t \in \mathcal{X}_z] \\
&\geq 0, \quad (\text{A.165})
\end{aligned}$$

where Eq. (A.165) holds by Eq. (12.22) and inductive hypothesis $\forall i > j : \mathbb{E}[\mathcal{G}(\xi_{t+h}) \mid \xi_t \in \mathcal{X}_i] \geq \mathbb{E}[\mathcal{G}(\xi_{t+h}) \mid \xi_t \in \mathcal{X}_j]$. Thus, the induction leads to

$$\forall h > 0, \forall i : \mathbb{E}[\mathcal{G}(\xi_{t+h}) \mid \xi_t \in \mathcal{X}_i] \geq \mathbb{E}[\mathcal{G}(\xi_{t+h}^F) \mid \xi_t^F \in \mathcal{X}_i^F], \quad (\text{A.166})$$

implying that, starting from subspace \mathcal{X}_i , EA produces solutions closer to the optimal solutions than that of EA^F all the time.

Next, we expand $P(\xi_0 \in \mathcal{X}_i) \mathbb{E}[\mathcal{G}(\xi_t) \mid \xi_0 \in \mathcal{X}_i]$ as

$$\begin{aligned}
& P(\xi_0 \in \mathcal{X}_i) \mathbb{E}[\mathcal{G}(\xi_t) \mid \xi_0 \in \mathcal{X}_i] \\
&= P(\xi_0 \in \mathcal{X}_i) \sum_{j=1}^m \mathcal{G}(\mathcal{X}_j) P(\xi_t \in \mathcal{X}_j \mid \xi_0 \in \mathcal{X}_i) \\
&= \sum_{\substack{j=1 \\ j \neq i}}^m \mathcal{G}(\mathcal{X}_j) P(\xi_t \in \mathcal{X}_j \mid \xi_0 \in \mathcal{X}_i) P(\xi_0 \in \mathcal{X}_i) \\
&\quad + \mathcal{G}(\mathcal{X}_i) P(\xi_t \in \mathcal{X}_i \mid \xi_0 \in \mathcal{X}_i) P(\xi_0 \in \mathcal{X}_i). \quad (\text{A.167})
\end{aligned}$$

Summing it over subspaces, we have

$$\begin{aligned}
& \sum_{i=1}^m P(\xi_0 \in \mathcal{X}_i) \mathbb{E}[\mathcal{G}(\xi_t) \mid \xi_0 \in \mathcal{X}_i] \\
&= \sum_{i=1}^m \left(\sum_{\substack{j=1 \\ j \neq i}}^m \mathcal{G}(\mathcal{X}_j) P(\xi_t \in \mathcal{X}_j \mid \xi_0 \in \mathcal{X}_i) P(\xi_0 \in \mathcal{X}_i) \right) \\
&\quad + \sum_{i=1}^m \mathcal{G}(\mathcal{X}_i) P(\xi_t \in \mathcal{X}_i \mid \xi_0 \in \mathcal{X}_i) P(\xi_0 \in \mathcal{X}_i) \\
&= \sum_{i=1}^m \mathcal{G}(\mathcal{X}_i) \sum_{\substack{j=1 \\ j \neq i}}^m P(\xi_t \in \mathcal{X}_i \mid \xi_0 \in \mathcal{X}_j) P(\xi_0 \in \mathcal{X}_j) \\
&\quad + \sum_{i=1}^m \mathcal{G}(\mathcal{X}_i) P(\xi_t \in \mathcal{X}_i \mid \xi_0 \in \mathcal{X}_i) P(\xi_0 \in \mathcal{X}_i) \\
&= \sum_{i=1}^m \mathcal{G}(\mathcal{X}_i) P(\xi_t \in \mathcal{X}_i),
\end{aligned} \tag{A.168}$$

and likewise,

$$\sum_{i=1}^m P(\xi_0^F \in \mathcal{X}_i^F) \mathbb{E}[\mathcal{G}(\xi_t^F) \mid \xi_0^F \in \mathcal{X}_i^F] = \sum_{i=1}^m \mathcal{G}(\mathcal{X}_i) P(\xi_t^F \in \mathcal{X}_i^F). \tag{A.169}$$

By Eq. (A.166) and $P(\xi_0 \in \mathcal{X}_i) = P(\xi_0^F \in \mathcal{X}_i^F)$, we have

$$\begin{aligned}
\forall t : & \sum_{i=1}^m P(\xi_0 \in \mathcal{X}_i) \mathbb{E}[\mathcal{G}(\xi_t) \mid \xi_0 \in \mathcal{X}_i] \\
& \geq \sum_{i=1}^m P(\xi_0^F \in \mathcal{X}_i^F) \mathbb{E}[\mathcal{G}(\xi_t^F) \mid \xi_0^F \in \mathcal{X}_i^F].
\end{aligned} \tag{A.170}$$

Combining Eqs. (A.168) to (A.170), we have

$$\forall t : \sum_{i=1}^m \mathcal{G}(\mathcal{X}_i) P(\xi_t \in \mathcal{X}_i) \geq \sum_{i=1}^m \mathcal{G}(\mathcal{X}_i) P(\xi_t^F \in \mathcal{X}_i^F). \tag{A.171}$$

Because

$$\begin{aligned}
P(\xi_{t+1} \in \mathcal{X}^*) &= P(\xi_t \in \mathcal{X}^*) + \sum_{x \notin \mathcal{X}^*} \mathcal{G}(x) P(\xi_t = x) \\
&= \mathcal{G}(\mathcal{X}^*) P(\xi_t \in \mathcal{X}^*) + \sum_{x \notin \mathcal{X}^*} \mathcal{G}(x) P(\xi_t = x)
\end{aligned}$$

$$= \sum_{i=1}^m \mathcal{G}(\mathcal{X}_i) P(\xi_t \in \mathcal{X}_i), \quad (\text{A.172})$$

and likewise $P(\xi_{t+1}^F \in \mathcal{X}^*) = \sum_{i=1}^m \mathcal{G}(\mathcal{X}_i) P(\xi_t^F \in \mathcal{X}_i^F)$, Eq. (A.171) leads to

$$\forall t : P(\xi_t \in \mathcal{X}^*) \geq P(\xi_t^F \in \mathcal{X}^*). \quad (\text{A.173})$$

Following the proof of Theorem 12.1, we have

$$\mathbb{E}[\tau^F \mid \xi_0^F \sim \pi_0^F] \geq \mathbb{E}[\tau \mid \xi_0 \sim \pi_0]. \quad (\text{A.174})$$

□

A.8 Proofs in Chapter 15

Proof of Theorem 15.3.

We first analyze $f(\mathbf{s})$. When $|\text{supp}(\mathbf{s})| = 1$, the solution $(0, i, 0, \dots, 0)$ with $i \in [k]$ has the largest f value $3 + 2/k$, which is calculated by $\mathbb{E}[a_2] = \mathbb{E}[a_4] = \mathbb{E}[a_5] = 1$, $\mathbb{E}[a_8] = \mathbb{E}[a_9] = 1/k$ and $\mathbb{E}[a_1] = \mathbb{E}[a_3] = \mathbb{E}[a_6] = \mathbb{E}[a_7] = 0$. When $s_2 > 0$ and $|\text{supp}(\mathbf{s})| = 2$, we have, $\forall i, j \in [k]$:

$$f((j, i, 0, \dots, 0)) = f((0, i, j, 0, \dots, 0)) = 5 + \frac{3}{k} - \frac{1}{k^2}, \quad (\text{A.175})$$

$$f((0, i, 0, j, 0, \dots, 0)) = f((0, i, 0, 0, j, 0, \dots, 0)) = 4 + \frac{1}{k}, \quad (\text{A.176})$$

$$\text{otherwise, } f(\mathbf{s}) \leq 4 + \frac{2}{k}. \quad (\text{A.177})$$

In the case of $b = 2$, it can be verified that $(i, 0, j, 0, \dots, 0)$ is a global optimal solution with objective value $6 + 2/k$. When $|\text{supp}(\mathbf{s})| = 3$, the solution $(i, j, l, 0, \dots, 0)$ has the largest f value $7 + 4/k - 2/k^2$.

Thus, the structure of f is similar to that for influence maximization. We can use the same proof as Theorem 15.2 to prove that POkSS finds a global optimal solution in $O(bn)$ expected number of iterations, whereas the generalized greedy algorithm will get trapped in a local optimal solution. □

Proof of Theorem 15.4.

By calculating $f(\mathbf{s})$ with $|\text{supp}(\mathbf{s})| \leq 3$, we can follow the solution path of the generalized greedy algorithm. When $|\text{supp}(\mathbf{s})| = 1$, $f((1, 0, \dots, 0)) = 1.5$, $f((0, 2, 0, \dots, 0)) = f((0, 0, 3, 0, \dots, 0)) = f((0, 0, 0, 4, 0, \dots, 0)) = 1$, and $f(\mathbf{s}) = 0$ otherwise. Thus, the generalized greedy algorithm first finds $(1, 0, \dots, 0)$. When $|\text{supp}(\mathbf{s})| = 2$ and $s_1 = 1$, we have $f((1, 2, 0, \dots, 0)) = f((1, 0, 3, 0, \dots, 0)) = f((1, 0, 0, 4, 0, \dots, 0)) = 2.16$, and $f(\mathbf{s}) = 1.5$ otherwise. Thus, the next solution can be any of these three solutions. As

$f((1, 2, 3, 0, \dots, 0)) = 2.5$ and $f((1, 2, 0, 4, 0, \dots, 0)) = f((1, 0, 3, 4, 0, \dots, 0)) = 2.75$, the generalized greedy algorithm will output the solution $s_{local} = (1, 2, 0, 4, 0, \dots, 0)$ or $(1, 0, 3, 4, 0, \dots, 0)$. It can be verified that the global optimal solution $s_{global} = (0, 2, 3, 4, 0, \dots, 0)$ with $f(s_{global}) = 3$. Thus, the generalized greedy algorithm cannot find s_{global} .

For POKSS, we can first use the same proof as Theorem 15.2 to derive that s_{local} will be found in $O(b)$ expected number of iterations, i.e., $\mathbb{E}[T_1] = O(b)$. Then, by selecting s_{local} in line 4 and using mutation in line 5, the solution $s^* = (1, 2, 3, 4, 0, \dots, 0)$ will be generated with probability at least $(1/(2b)) \cdot (1/n)(1 - 1/n)^{n-1}(1/k) \geq 1/(2ekbn)$. Because $f(s^*)$ reaches the largest f value 3 and we pessimistically assume that s_{global} has not been found, no solution in P can dominate s^* and it will go into the SLS procedure, which quickly finds s_{global} by letting $s_1^* = 0$ with probability at least $1 - 1/e$. Thus, $\mathbb{E}[T_2] \leq 2ekbn \cdot e/(e - 1)$. By combining these two phases, the expected number of iterations for POKSS finding s_{global} is at most $\mathbb{E}[T_1] + \mathbb{E}[T_2] = O(kbn)$. \square

A.9 Proofs in Chapter 17

Proof of Lemma 17.1.

Let s^* be an optimal subset, i.e., $f(s^*) = \text{OPT}$. Let $\hat{v} = \arg \max_{v \in s^* \setminus s} f^n(s \cup v)$. Then, we have

$$f(s^*) - f(s) \leq f(s^* \cup s) - f(s) \quad (\text{A.178})$$

$$\leq \frac{1}{\gamma_{s,b}} \sum_{v \in s^* \setminus s} (f(s \cup v) - f(s)) \quad (\text{A.179})$$

$$\leq \frac{1}{\gamma_{s,b}} \sum_{v \in s^* \setminus s} \left(\frac{1}{1-\epsilon} f^n(s \cup v) - f(s) \right) \quad (\text{A.180})$$

$$\leq \frac{b}{\gamma_{s,b}} \left(\frac{1}{1-\epsilon} f^n(s \cup \hat{v}) - f(s) \right), \quad (\text{A.181})$$

where Eq. (A.178) holds by the monotonicity of f , Eq. (A.179) holds by the definition of submodularity ratio and $|s^*| \leq b$, and Eq. (A.180) holds by $f^n(s) \geq (1 - \epsilon)f(s)$. By a simple transformation, we can equivalently get

$$f^n(s \cup \hat{v}) \geq (1 - \epsilon) \left(\left(1 - \frac{\gamma_{s,b}}{b}\right) f(s) + \frac{\gamma_{s,b}}{b} \cdot \text{OPT} \right). \quad (\text{A.182})$$

By applying $f(s) \geq f^n(s)/(1 + \epsilon)$ to Eq. (A.182), the lemma holds. \square

Proof of Theorem 17.5.

Let $A = \{S_1, \dots, S_l\}$ and $B = \{S_{l+1}, \dots, S_{2l}\}$. For the greedy algorithm, if without noise, it first selects one S_i from A , and continues to select S_i from

B until reaching the budget. Thus, the greedy algorithm finds an optimal solution. In the presence of noise, after selecting one S_i from A , it continues to select S_i from A rather than from B , because $\forall S \subseteq A, S_i \in B : f^n(S) = 2 + \delta > 2 = f^n(S \cup \{S_i\})$. Thus, the approximation ratio is only $2/(b+1)$.

For POSS under noise, we show that it can efficiently follow the path $0^n \rightarrow \{S^*\} \rightarrow \{S^*\} \cup B_2 \rightarrow \{S^*\} \cup B_3 \rightarrow \dots \rightarrow \{S^*\} \cup B_{b-1}$, i.e., an optimal solution, where S^* denotes any element from A and B_i denotes any subset of B with size i . Note that the solutions on the path will always be kept in the population P once found, because no other solution can dominate them. The probability of the first “ \rightarrow ” on the path is at least $(1/P_{\max}) \cdot (l/n)(1-1/n)^{n-1}$, because it is sufficient to select 0^n in line 4 of Algorithm 14.2, flip one of its first l 0-bits and keep the other bits unchanged in line 5. For the second “ \rightarrow ”, which performs multi-bit search, the probability is at least $(1/P_{\max}) \cdot ((l \choose 2)/n^2)(1-1/n)^{n-2}$, because it is sufficient to select $\{S^*\}$ and flip any two 0-bits in its second half. For the i -th “ \rightarrow ” with $3 \leq i \leq b-1$, the probability is at least $(1/P_{\max}) \cdot ((l-i+1)/n)(1-1/n)^{n-1}$, because it is sufficient to select the left solution of “ \rightarrow ” and flip only one 0-bit in its second half. Thus, starting from 0^n , POSS can follow the path in

$$enP_{\max} \cdot \left(\frac{1}{l} + \frac{4}{l-1} + \sum_{i=3}^{b-1} \frac{1}{l-i+1} \right) = O(nP_{\max} \log n) \quad (\text{A.183})$$

expected number of iterations. As $P_{\max} \leq 2b$, the number of iterations for finding an optimal solution is $O(bn \log n)$ in expectation. \square

Proof of Theorem 17.6.

For the greedy algorithm, if without noise, it first selects S_{4l-2} because $|S_{4l-2}|$ is the largest, and then finds the optimal solution $\{S_{4l-2}, S_{4l-1}\}$. In the presence of noise, S_{4l} is first selected because $f^n(\{S_{4l}\}) = 2l$ is the largest, and then the solution $\{S_{4l}, S_{4l-1}\}$ is found. Thus, the approximation ratio is only $(3l-2)/(4l-3)$.

For POSS under noise, it can efficiently follow the path $0^n \rightarrow \{S_{4l}\} \rightarrow \{S_{4l}, S_{4l-1}\} \rightarrow \{S_{4l-2}, S_{4l-1}, *\}$, where $*$ denotes any subset S_i with $i \neq 4l-2, 4l-1$. In this procedure, we can pessimistically assume that the optimal solution $\{S_{4l-2}, S_{4l-1}\}$ will never be found, because the goal is to derive a running time upper bound for finding it. Note that the solutions on the path will always be kept in P once found, because no other solution can dominate them. The probability of “ \rightarrow ” is at least $(1/P_{\max}) \cdot (1/n)(1-1/n)^{n-1} \geq 1/(enP_{\max})$, because it is sufficient to select the solution on the left of “ \rightarrow ” and flip only one specific 0-bit. Thus, starting from 0^n , POSS can follow the path in $3 \cdot enP_{\max}$ expected number of iterations. After that, the optimal solution $\{S_{4l-2}, S_{4l-1}\}$ can be found by selecting $\{S_{4l-2}, S_{4l-1}, *\}$ and flipping a specific 1-bit, which performs backward search, and occurs with probability at least $1/(enP_{\max})$. Thus, the total number of required iterations is at most $4enP_{\max}$ in expectation. As $P_{\max} \leq 4$, we have $\mathbb{E}[T] = O(n)$. \square

References

- A. N. Aizawa and B. W. Wah. Scheduling of genetic algorithms in a noisy environment. *Evolutionary Computation*, 2(2):97–122, 1994.
- Y. Akimoto, S. Astete-Morales, and O. Teytaud. Analysis of runtime of optimization algorithms for noisy functions over discrete codomains. *Theoretical Computer Science*, 605:42–50, 2015.
- R. Aleliunas, R. Karp, R. Lipton, L. Lovasz, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 218–223, San Juan, Puerto Rico, 1979.
- N. Alon, D. Moshkovitz, and S. Safra. Algorithmic construction of sets for k -restrictions. *ACM Transactions on Algorithms*, 2(2):153–177, 2006.
- D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *Proceedings of the 4th International Workshop on Ambient Assisted Living and Home Care (IWAAL)*, pages 216–223, Vitoria-Gasteiz, Spain, 2012.
- A. Arias-Montano, C. A. Coello Coello, and E. Mezura-Montes. Multiobjective evolutionary algorithms in aeronautical and aerospace engineering. *IEEE Transactions on Evolutionary Computation*, 16(5):662–694, 2012.
- D. V. Arnold and H.-G. Beyer. A general noise model and its effects on evolution strategy performance. *IEEE Transactions on Evolutionary Computation*, 10(4):380–391, 2006.
- A. Auger and B. Doerr. *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. World Scientific, Singapore, 2011.
- T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK, 1996.
- G. Badkobeh, P. K. Lehre, and D. Sudholt. Unbiased black-box complexity of parallel search. In *Proceedings of the 13th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 892–901, Ljubljana, Slovenia, 2014.
- W. Bai, R. Iyer, K. Wei, and J. Bilmes. Algorithms for optimizing the ratio of submodular functions. In *Proceedings of the 33rd International Conference on Machine*

- Learning (ICML)*, pages 2751–2759, New York, NY, 2016.
- R. E. Banfield, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. Ensemble diversity measures and their application to thinning. *Information Fusion*, 6(1):49–62, 2005.
- N. Barbieri, F. Bonchi, and G. Manco. Topic-aware social influence propagation models. In *Proceedings of the 12th IEEE International Conference on Data Mining (ICDM)*, pages 81–90, Brussels, Belgium, 2012.
- T. Bartz-Beielstein. Evolution strategies and threshold selection. In *Proceedings of the 2nd International Workshop on Hybrid Metaheuristics (HM)*, pages 104–115, Barcelona, Spain, 2005.
- T. Bartz-Beielstein and S. Markon. Threshold selection, hypothesis tests, and DOE methods. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 777–782, Honolulu, HI, 2002.
- A. Ben Hadj-Alouane and J. C. Bean. A genetic algorithm for the multiple-choice integer program. *Operations Research*, 45(1):92–101, 1997.
- E. Benkhelifa, G. Dragffy, A. Pipe, and M. Nibouche. Design innovation for real world applications, using evolutionary algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 918–924, Trondheim, Norway, 2009.
- D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Cambridge, MA, 1999.
- H.-G. Beyer. Evolutionary algorithms in noisy environments: Theoretical issues and guidelines for practice. *Computer Methods in Applied Mechanics and Engineering*, 186(2):239–267, 2000.
- A. A. Bian, J. M. Buhmann, A. Krause, and S. Tschiatschek. Guarantees for greedy maximization of non-submodular functions with applications. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 498–507, Sydney, Australia, 2017.
- S. Böttcher, B. Doerr, and F. Neumann. Optimal fixed and adaptive mutation rates for the leadingones problem. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 1–10, Krakow, Poland, 2010.
- J. Branke and C. Schmidt. Selection in the presence of noise. In *Proceedings of the 5th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 766–777, Chicago, IL, 2003.
- J. Branke and C. Schmidt. Sequential sampling in noisy environments. In *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 202–211, Birmingham, UK, 2004.
- L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
- G. Brown, J. Wyatt, R. Harris, and X. Yao. Diversity creation methods: A survey and categorisation. *Information Fusion*, 6(1):5–20, 2005.
- Z. Cai and Y. Wang. A multiobjective optimization-based evolutionary algorithm for constrained optimization. *IEEE Transactions on Evolutionary Computation*, 10(6):658–675, 2006.
- R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, pages 18–25, Banff, Canada, 2004.

- P. D. Castro, G. P. Coelho, M. F. Caetano, and F. J. Von Zuben. Designing ensembles of fuzzy classification systems: An immune-inspired approach. In *Proceedings of the 4th International Conference on Artificial Immune Systems (ICARIS)*, pages 469–482, Banff, Canada, 2005.
- H. Chen and X. Yao. Multiobjective neural network ensembles based on regularized negative correlation learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(12):1738–1751, 2010.
- T. Chen, J. He, G. Sun, G. Chen, and X. Yao. A new approach for analyzing average time complexity of population-based evolutionary algorithms on unimodal problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 39(5):1092–1106, 2009a.
- T. Chen, J. He, G. Chen, and X. Yao. Choosing selection pressure for wide-gap problems. *Theoretical Computer Science*, 411(6):926–934, 2010a.
- T. Chen, K. Tang, G. Chen, and X. Yao. A large population size can be unhelpful in evolutionary algorithms. *Theoretical Computer Science*, 436(8):54–70, 2012.
- W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 199–208, Paris, France, 2009b.
- W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1029–1038, Washington, DC, 2010b.
- Y. Chen, H. Hassani, A. Karbasi, and A. Krause. Sequential information maximization: When is greedy near-optimal? In *Proceedings of the 28th Annual Conference on Learning Theory (COLT)*, pages 338–363, Paris, France, 2015.
- V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.
- C. A. Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11):1245–1287, 2002.
- D. W. Coit and A. E. Smith. Penalty guided genetic search for reliability design optimization. *Computers and Industrial Engineering*, 30(4):895–904, 1996.
- M. Conforti and G. Cornuéjols. Submodular set functions, matroids and the greedy algorithm: Tight worst-case bounds and some generalizations of the Rado-Edmonds theorem. *Discrete Applied Mathematics*, 7(3):251–274, 1984.
- T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2001.
- N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, Cambridge, UK, 2000.
- D.-C. Dang and P. K. Lehre. Efficient optimisation of noisy fitness functions with population-based evolutionary algorithms. In *Proceedings of the 13th International Workshop on Foundations of Genetic Algorithms (FOGA)*, pages 62–68, Aberystwyth, UK, 2015.
- D.-C. Dang and P. K. Lehre. Self-adaptation of mutation rates in non-elitist populations. In *Proceedings of the 14th International Conference on Parallel Problem*

- Solving from Nature (PPSN)*, pages 803–813, Edinburgh, Scotland, 2016.
- A. Das and D. Kempe. Algorithms for subset selection in linear regression. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 45–54, Victoria, Canada, 2008.
- A. Das and D. Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 1057–1064, Bellevue, WA, 2011.
- G. Davis, S. Mallat, and M. Avellaneda. Adaptive greedy approximations. *Constructive Approximation*, 13(1):57–98, 1997.
- K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2):311–338, 2000.
- K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, New York, NY, 2001.
- K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
- J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.
- G. Diekhoff. *Statistics for the Social and Behavioral Sciences: Univariate, Bivariate, Multivariate*. William C Brown Pub, Dubuque, IA, 1992.
- T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.
- M. Dietzfelbinger, B. Naudts, C. Van Hoyweghen, and I. Wegener. The analysis of a recombinative hill-climber on H-IFF. *IEEE Transactions on Evolutionary Computation*, 7(5):417–423, 2003.
- B. Doerr and L. A. Goldberg. Adaptive drift analysis. *Algorithmica*, 65(1):224–250, 2013.
- B. Doerr and M. Künemann. Optimizing linear functions with the $(1+\lambda)$ evolutionary algorithm - different asymptotic runtimes for different instances. *Theoretical Computer Science*, 561:3–23, 2015.
- B. Doerr and S. Pohl. Run-time analysis of the $(1+1)$ evolutionary algorithm optimizing linear functions over a finite alphabet. In *Proceedings of the 14th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1317–1324, New York, NY, 2012.
- B. Doerr and M. Theile. Improved analysis methods for crossover-based algorithms. In *Proceedings of the 11th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 247–254, Montreal, Canada, 2009.
- B. Doerr, N. Hebbinghaus, and F. Neumann. Speeding up evolutionary algorithms through restricted mutation operators. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 978–987, Reykjavik, Iceland, 2006.
- B. Doerr, E. Happ, and C. Klein. Crossover can provably be useful in evolutionary computation. In *Proceedings of the 10th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 539–546, Atlanta, GA, 2008a.

- B. Doerr, T. Jansen, and C. Klein. Comparing global and local mutations on bit strings. In *Proceedings of the 10th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 929–936, Atlanta, GA, 2008b.
- B. Doerr, D. Johannsen, T. Kötzing, F. Neumann, and M. Theile. More effective crossover operators for the all-pairs shortest path problem. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 184–193, Krakow, Poland, 2010a.
- B. Doerr, D. Johannsen, and C. Winzen. Drift analysis and linear functions revisited. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, Barcelona, Spain, 2010b.
- B. Doerr, A. Eremeev, F. Neumann, M. Theile, and C. Thyssen. Evolutionary algorithms and dynamic programming. *Theoretical Computer Science*, 412(43):6020–6035, 2011a.
- B. Doerr, D. Johannsen, and M. Schmidt. Runtime analysis of the (1+1) evolutionary algorithm on strings over finite alphabets. In *Proceedings of the 11th International Workshop on Foundations of Genetic Algorithms (FOGA)*, pages 119–126, Schwarzenberg, Austria, 2011b.
- B. Doerr, A. Hota, and T. Kötzing. Ants easily solve stochastic shortest path problems. In *Proceedings of the 14th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 17–24, Philadelphia, PA, 2012a.
- B. Doerr, D. Johannsen, and C. Winzen. Non-existence of linear universal drift functions. *Theoretical Computer Science*, 436:71–86, 2012b.
- B. Doerr, D. Johannsen, and C. Winzen. Multiplicative drift analysis. *Algorithmica*, 64(4):673–697, 2012c.
- B. Doerr, C. Doerr, and F. Ebel. Lessons from the black-box: Fast crossover-based genetic algorithms. In *Proceedings of the 15th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 781–788, Amsterdam, The Netherlands, 2013a.
- B. Doerr, T. Jansen, D. Sudholt, C. Winzen, and C. Zarges. Mutation rate matters even when optimizing monotonic functions. *Evolutionary Computation*, 21(1):1–27, 2013b.
- B. Doerr, H. P. Le, R. Makhmara, and T. D. Nguyen. Fast genetic algorithms. In *Proceedings of the 19th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 777–784, Berlin, Germany, 2017.
- B. Doerr, C. Witt, and J. Yang. Runtime analysis for self-adaptive mutation rates. In *Proceedings of the 20th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1475–1482, Kyoto, Japan, 2018.
- P. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.
- S. Droste. Analysis of the (1+1) EA for a noisy OneMax. In *Proceedings of the 6th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1088–1099, Seattle, WA, 2004.
- S. Droste, T. Jansen, and I. Wegener. A rigorous complexity analysis of the (1+1) evolutionary algorithm for linear functions with Boolean inputs. *Evolutionary Computation*, 6(2):185–196, 1998.

- S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276(1-2):51–81, 2002.
- G. Durrett, F. Neumann, and U.-M. O'Reilly. Computational complexity analysis of simple genetic programming on two problems modeling isolated program semantics. In *Proceedings of the 11th International Workshop on Foundations of Genetic Algorithms (FOGA)*, pages 69–80, Schwarzenberg, Austria, 2011.
- J. Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1(1): 127–136, 1971.
- A.-E. Eiben, P.-E. Raué, and Z. Ruttkay. Genetic algorithms with multi-parent recombination. In *Proceedings of the 3rd International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 78–87, Jerusalem, Israel, 1994.
- E. R. Elenberg, R. Khanna, A. G. Dimakis, and S. Negahban. Restricted strong convexity implies weak submodularity. *Annals of Statistics*, 46(6B):3539–3568, 2018.
- C. Erbas, S. Cerav-Erbas, and A. D. Pimentel. Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. *IEEE Transactions on Evolutionary Computation*, 10(3): 358–374, 2006.
- J. Fan and R. Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 96(456):1348–1360, 2001.
- U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4): 634–652, 1998.
- M. Feldmann and T. Kötzing. Optimizing expected path lengths with ant colony optimization using fitness proportional update. In *Proceedings of the 12th International Workshop on Foundations of Genetic Algorithms (FOGA)*, pages 65–74, Adelaide, Australia, 2013.
- S. Fischer and I. Wegener. The one-dimensional Ising model: Mutation versus recombination. *Theoretical Computer Science*, 344(2-3):208–225, 2005.
- J. M. Fitzpatrick and J. J. Grefenstette. Genetic algorithms in noisy environments. *Machine Learning*, 3(2-3):101–120, 1988.
- J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, New York, NY, 2006.
- H. Fournier and O. Teytaud. Lower bounds for comparison based evolution strategies using VC-dimension and sign patterns. *Algorithmica*, 59(3):387–408, 2011.
- T. Friedrich and F. Neumann. Maximizing submodular functions under matroid constraints by evolutionary algorithms. *Evolutionary Computation*, 23(4):543–558, 2015.
- T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt. On improving approximate solutions by evolutionary algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 2614–2621, Singapore, 2007.
- T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt. Approximating covering problems by randomized search heuristics using multi-objective models. *Evolutionary Computation*, 18(4):617–633, 2010.
- T. Friedrich, T. Kötzing, M. S. Krejca, and A. M. Sutton. Robustness of ant colony optimization to noise. *Evolutionary Computation*, 24(2):237–254, 2016.

- T. Friedrich, T. Kötzing, M. S. Krejca, and A. M. Sutton. The compact genetic algorithm is efficient under extreme Gaussian noise. *IEEE Transactions on Evolutionary Computation*, 21(3):477–490, 2017.
- S. Fujishige and S. Iwata. Bisubmodular function minimization. *SIAM Journal on Discrete Mathematics*, 19(4):1065–1073, 2005.
- G. Giacinto, F. Roli, and G. Fumera. Design of effective multiple classifier systems by clustering of classifiers. In *Proceedings of the 15th International Conference on Pattern Recognition (ICPR)*, pages 160–163, Barcelona, Spain, 2000.
- O. Giel. Expected runtimes of a simple multi-objective evolutionary algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1918–1925, Canberra, Australia, 2003.
- O. Giel and I. Wegener. Evolutionary algorithms and the maximum matching problem. Technical Report CI-142/02, Department of Computer Science, University of Dortmund, Dortmund, Germany, 2002.
- O. Giel and I. Wegener. Evolutionary algorithms and the maximum matching problem. In *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 415–426, Berlin, Germany, 2003.
- O. Giel and I. Wegener. Maximum cardinality matchings on trees by randomized local search. In *Proceedings of the 8th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 539–546, Seattle, WA, 2006.
- C. Gießen and T. Kötzing. Robustness of populations in stochastic environments. *Algorithmica*, 75(3):462–489, 2016.
- C. Gießen and C. Witt. Population size vs. mutation strength for the $(1+\lambda)$ EA on OneMax. In *Proceedings of the 17th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1439–1446, Madrid, Spain, 2015.
- D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
- A. Goyal, W. Lu, and L. V. Lakshmanan. Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *Proceedings of the 11th IEEE International Conference on Data Mining (ICDM)*, pages 211–220, Vancouver, Canada, 2011.
- E. J. Gumbel. The maxima of the mean largest value and of the range. *Annals of Mathematical Statistics*, 25:76–84, 1954.
- I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.
- B. Hajek. Hitting-time and occupation-time bounds implied by drift analysis with applications. *Advances in Applied Probability*, 14(3):502–525, 1982.
- T. Hanne. On the convergence of multiobjective evolutionary algorithms. *European Journal of Operational Research*, 117(3):553–564, 1999.
- E. Happ, D. Johannsen, C. Klein, and F. Neumann. Rigorous analyses of fitness-proportional selection for optimizing linear functions. In *Proceedings of the 10th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 953–960, Atlanta, GA, 2008.
- H. O. Hartley and H. A. David. Universal bounds for mean range and extreme observations. *Annals of Mathematical Statistics*, 25:85–99, 1954.

- A. Hassidim and Y. Singer. Submodular optimization under noise. In *Proceedings of the 30th Annual Conference on Learning Theory (COLT)*, pages 1069–1122, Amsterdam, The Netherlands, 2017.
- S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice-Hall, Upper Saddle River, NJ, 2nd edition, 1998.
- J. He and L. Kang. On the convergence rates of genetic algorithms. *Theoretical Computer Science*, 229(1-2):23–39, 1999.
- J. He and X. Yao. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, 127(1):57–85, 2001.
- J. He and X. Yao. From an individual to a population: An analysis of the first hitting time of population-based evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):495–511, 2002.
- J. He and X. Yao. Towards an analytic framework for analysing the computation time of evolutionary algorithms. *Artificial Intelligence*, 145(1-2):59–97, 2003a.
- J. He and X. Yao. An analysis of evolutionary algorithms for finding approximation solutions to hard optimisation problems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 2004–2010, Canberra, Australia, 2003b.
- J. He and X. Yao. A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3(1):21–35, 2004.
- J. He and X. Yu. Conditions for the convergence of evolutionary algorithms. *Journal of Systems Architecture*, 47(7):601–612, 2001.
- J. He and Y. Zhou. A comparison of GAs using penalizing infeasible solutions and repairing infeasible solutions on restrictive capacity knapsack problem. In *Proceedings of the 9th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, page 1518, London, UK, 2007.
- D. Hernández-Lobato, G. Martínez-Muñoz, and A. Suárez. Empirical analysis and evaluation of approximate techniques for pruning regression bagging ensembles. *Neurocomputing*, 74(12):2250–2264, 2011.
- T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, N. Salami, N. Kajihara, and N. Otsu. Real-world applications of analog and digital evolvable hardware. *IEEE Transactions on Evolutionary Computation*, 3(3):220–235, 1999.
- N. Hoai, R. McKay, and D. Essam. Representation and structural difficulty in genetic programming. *IEEE Transactions on Evolutionary Computation*, 10(2):157–166, 2006.
- T. Hogg and K. Lerman. Social dynamics of digg. *EPJ Data Science*, 1(5):1–26, 2012.
- A. Homaifar, C. X. Qi, and S. H. Lai. Constrained optimization via genetic algorithms. *Simulation*, 62(4):242–253, 1994.
- H. H. Hoos and T. Stützle. Towards a characterisation of the behaviour of stochastic local search algorithms for SAT. *Artificial Intelligence*, 112(1):213–232, 1999.
- H. H. Hoos and T. Stützle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.
- H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, San Francisco, CA, 2005.

- T. Horel and Y. Singer. Maximization of approximately submodular functions. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances In Neural Information Processing Systems 29 (NIPS)*, pages 3045–3053. Curran Associates, Inc., Red Hook, NY, 2016.
- G. S. Hornby, A. Globus, D. S. Linden, and J. D. Lohn. Automated antenna design with evolutionary algorithms. In *Proceedings of the American Institute of Aeronautics and Astronautics Conference on Space (AIAA Space)*, pages 19–21, San Jose, CA, 2006.
- S.-J. Huang, R. Jin, and Z.-H. Zhou. Active learning by querying informative and representative examples. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(10):1936–1949, 2014.
- A. Huber and V. Kolmogorov. Towards minimizing k -submodular functions. In *Proceedings of the 2nd International Symposium on Combinatorial Optimization (ISCO)*, pages 451–462, Athens, Greece, 2012.
- S. Iwata, S. Tanigawa, and Y. Yoshida. Improved approximation algorithms for k -submodular function maximization. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 404–413, Arlington, VA, 2016.
- R. Iyer and J. Bilmes. Algorithms for approximate minimization of the difference between submodular functions, with applications. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 407–417, Catalina Island, CA, 2012.
- R. Iyer and J. Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. In C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26 (NIPS)*, pages 2436–2444. Curran Associates, Inc., Red Hook, NY, 2013.
- R. Iyer, S. Jegelka, and J. Bilmes. Curvature and optimal algorithms for learning and minimizing submodular functions. In C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26 (NIPS)*, pages 2742–2750. Curran Associates, Inc., Red Hook, NY, 2013.
- J. Jägersküpper. A blend of Markov-chain and drift analysis. In *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 41–51, Dortmund, Germany, 2008.
- J. Jägersküpper and T. Storch. When the plus strategy outperforms the comma strategy and when not. In *Proceedings of the IEEE Symposium on Foundations of Computational Intelligence (FOCI)*, pages 25–32, Honolulu, HI, 2007.
- A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Upper Saddle River, NJ, 1988.
- T. Jansen and K. De Jong. An analysis of the role of offspring population size in EAs. In *Proceedings of the 4th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 238–246, New York, NY, 2002.
- T. Jansen and D. Sudholt. Analysis of an asymmetric mutation operator. *Evolutionary Computation*, 18(1):1–26, 2010.
- T. Jansen and I. Wegener. On the utility of populations in evolutionary algorithms. In *Proceedings of the 3rd ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1034–1041, San Francisco, CA, 2001.

- T. Jansen and I. Wegener. Real royal road functions-where crossover provably is essential. *Discrete Applied Mathematics*, 149(1-3):111–125, 2005.
- T. Jansen and I. Wegener. On the analysis of a dynamic evolutionary algorithm. *Journal of Discrete Algorithms*, 4(1):181–199, 2006.
- T. Jansen and I. Wegener. A comparison of simulated annealing with a simple evolutionary algorithm on pseudo-Boolean functions of unitation. *Theoretical Computer Science*, 386(1-2):73–93, 2007.
- Y. Jin and J. Branke. Evolutionary optimization in uncertain environments-A survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.
- D. Johannsen, P. Kurur, and J. Lengler. Can quantum search accelerate evolutionary algorithms? In *Proceedings of the 12th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1433–1440, Portland, OR, 2010.
- R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, Upper Saddle River, NJ, 6th edition, 2007.
- D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 137–146, Washington, DC, 2003.
- S. Khan, A. R. Baig, A. Ali, B. Haider, F. A. Khan, M. Y. Durrani, and M. Ishtiaq. Unordered rule discovery using ant colony optimization. *Science China Information Sciences*, 57(9):1–15, 2014.
- S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, 34(5):975–986, 1984.
- M. P. Kleeman, B. A. Seibert, G. B. Lamont, K. M. Hopkinson, and S. R. Graham. Solving multicommodity capacitated network design problems using multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 16(4):449–471, 2012.
- B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer-Verlag, Berlin, Germany, 2012.
- T. Kötzing, F. Neumann, and R. Spöhel. PAC learning and genetic programming. In *Proceedings of the 13th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 2091–2096, Dublin, Ireland, 2011a.
- T. Kötzing, D. Sudholt, and M. Theile. How crossover helps in pseudo-Boolean optimization. In *Proceedings of the 13th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 989–996, Dublin, Ireland, 2011b.
- T. Kötzing, A. M. Sutton, F. Neumann, and U.-M. O'Reilly. The max problem revisited: The importance of mutation in genetic programming. *Theoretical Computer Science*, 545:94–107, 2014.
- J. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, 1994.
- J. Koza. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3):251–284, 2010.
- J. Koza, M. A. Keane, and M. J. Streeter. What's AI done for me lately? Genetic programming's human-competitive results. *IEEE Intelligent Systems*, 18(3):25–31, 2003.

- S. Koziel and Z. Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19–44, 1999.
- A. Krause, B. McMahan, C. Guestrin, and A. Gupta. Robust submodular observation selection. *Journal of Machine Learning Research*, 9(12):2761–2801, 2008a.
- A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9:235–284, 2008b.
- M. Laumanns, L. Thiele, and E. Zitzler. Running time analysis of multiobjective evolutionary algorithms on pseudo-Boolean functions. *IEEE Transactions on Evolutionary Computation*, 8(2):170–182, 2004.
- A. Lazarevic and Z. Obradovic. Effective pruning of neural network classifier ensembles. In *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks (IJCNN)*, pages 796–801, Washington, DC, 2001.
- P. K. Lehre and C. Witt. Black-box search by unbiased variation. *Algorithmica*, 64(4): 623–642, 2012.
- P. K. Lehre and X. Yao. Crossover can be constructive when computing unique input output sequences. In *Proceedings of the 7th International Conference on Simulated Evolution and Learning (SEAL)*, pages 595–604, Melbourne, Australia, 2008.
- P. K. Lehre and X. Yao. On the impact of mutation-selection balance on the runtime of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 16 (2):225–241, 2012.
- N. Li and Z.-H. Zhou. Selective ensemble under regularization framework. In *Proceedings of the 8th International Workshop on Multiple Classifier Systems (MCS)*, pages 293–303, Reykjavik, Iceland, 2009.
- N. Li, Y. Yu, and Z.-H. Zhou. Diversity regularized ensemble pruning. In *Proceedings of the 23rd European Conference on Machine Learning (ECML)*, pages 330–345, Bristol, UK, 2012.
- S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.
- A. Lissavoi and P. S. Oliveto. On the time and space complexity of genetic programming for evolving Boolean conjunctions. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, pages 1363–1370, New Orleans, LA, 2018.
- H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer, Norwell, MA, 1998.
- D. D. Margineantu and T. G. Dietterich. Pruning adaptive boosting. In *Proceedings of the 14th International Conference on Machine Learning (ICML)*, pages 211–218, Nashville, TN, 1997.
- S. Markon, D. V. Arnold, T. Bäck, T. Bartz-Beielstein, and H.-G. Beyer. Thresholding—A selection operator for noisy ES. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 465–472, Seoul, South Korea, 2001.
- A. W. Marshall, I. Olkin, and B. Arnold. *Inequalities: Theory of Majorization and Its Applications*. Springer, New York, NY, 2nd edition, 2011.
- G. Martínez-Muñoz, D. Hernández-Lobato, and A. Suárez. An analysis of ensemble pruning techniques based on ordered aggregation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):245–259, 2009.

- S. T. McCormick and S. Fujishige. Strongly polynomial and fully combinatorial algorithms for bisubmodular function minimization. *Mathematical Programming*, 122(1):87–120, 2010.
- G. J. McLachlan. *Discriminant Analysis and Statistical Pattern Recognition*. Wiley, New York, NY, 1992.
- O. J. Mengshoel. Understanding the role of noise in stochastic local search: Analysis and experiments. *Artificial Intelligence*, 172(8):955–990, 2008.
- J. Mestre. Greedy in approximation algorithms. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, pages 528–539, Zurich, Switzerland, 2006.
- E. Mezura-Montes and C. A. Coello Coello. A simple multimembered evolution strategy to solve constrained optimization problems. *IEEE Transactions on Evolutionary Computation*, 9(1):1–17, 2005.
- Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.
- A. Miller. *Subset Selection in Regression*. Chapman and Hall/CRC, London, UK, 2nd edition, 2002.
- M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. *Optimization Techniques*, 7:234–243, 1978.
- B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrák, and A. Krause. Lazier than lazy greedy. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1812–1818, Austin, TX, 2015.
- T. Mitchell. *Machine Learning*. McGraw Hill, New York, NY, 1997.
- M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. MIT press, London, UK, 2012.
- A. Mukhopadhyay, U. Maulik, S. Bandyopadhyay, and C. A. Coello Coello. A survey of multiobjective evolutionary algorithms for data mining: Part I. *IEEE Transactions on Evolutionary Computation*, 18(1):4–19, 2014a.
- A. Mukhopadhyay, U. Maulik, S. Bandyopadhyay, and C. A. Coello Coello. Survey of multiobjective evolutionary algorithms for data mining: Part II. *IEEE Transactions on Evolutionary Computation*, 18(1):20–35, 2014b.
- B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24(2):227–234, 1995.
- G. L. Nemhauser and L. A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978.
- G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions – I. *Mathematical Programming*, 14(1):265–294, 1978.
- F. Neumann. Computational complexity analysis of multi-objective genetic programming. In *Proceedings of the 14th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 799–806, Philadelphia, PA, 2012.
- F. Neumann and M. Laumanns. Speeding up approximation algorithms for NP-hard spanning forest problems by multi-objective optimization. In *Proceedings of the 7th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 745–

- 756, Valdivia, Chile, 2006.
- F. Neumann and J. Reichel. Approximating minimum multicuts by evolutionary multi-objective algorithms. In *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 72–81, Dortmund, Germany, 2008.
- F. Neumann and M. Theile. How crossover speeds up evolutionary algorithms for the multi-criteria all-pairs-shortest-path problem. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 667–676, Krakow, Poland, 2010.
- F. Neumann and I. Wegener. Minimum spanning trees made easier via multi-objective optimization. *Natural Computing*, 5(3):305–319, 2006.
- F. Neumann and I. Wegener. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science*, 378(1):32–40, 2007.
- F. Neumann and C. Witt. *Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity*. Springer, Berlin, Germany, 2010.
- F. Neumann, P. S. Oliveto, and C. Witt. Theoretical analysis of fitness-proportional selection: Landscapes and efficiency. In *Proceedings of the 11th ACM Annual conference on Genetic and Evolutionary Computation (GECCO)*, pages 835–842, Montreal, Canada, 2009.
- F. Neumann, J. Reichel, and M. Skutella. Computing minimum cuts by randomized search heuristics. *Algorithmica*, 59(3):323–342, 2011.
- A. Nguyen, T. Urli, and M. Wagner. Single-and multi-objective genetic programming: New bounds for weighted order and majority. In *Proceedings of the 12th International Workshop on Foundations of Genetic Algorithms (FOGA)*, pages 161–172, Adelaide, Australia, 2013.
- J. R. Norris. *Markov Chains*. Cambridge University Press, Cambridge, UK, 1997.
- N. Ohsaka and Y. Yoshida. Monotone k -submodular function maximization with size constraints. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28 (NIPS)*, pages 694–702. Curran Associates, Inc., Red Hook, NY, 2015.
- P. S. Oliveto and C. Witt. Simplified drift analysis for proving lower bounds in evolutionary computation. *Algorithmica*, 59(3):369–386, 2011.
- P. S. Oliveto and C. Witt. Erratum: Simplified drift analysis for proving lower bounds in evolutionary computation. *CORR* abs/1211.7184, 2012.
- P. S. Oliveto and C. Witt. On the runtime analysis of the simple genetic algorithm. *Theoretical Computer Science*, 545:2–19, 2014.
- P. S. Oliveto and C. Witt. Improved time complexity analysis of the simple genetic algorithm. *Theoretical Computer Science*, 605:21–41, 2015.
- P. S. Oliveto, J. He, and X. Yao. Analysis of population-based evolutionary algorithms for the vertex cover problem. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1563–1570, Hongkong, China, 2008.
- P. S. Oliveto, J. He, and X. Yao. Analysis of the (1+1)-EA for finding approximate solutions to vertex cover problems. *IEEE Transactions on Evolutionary Computation*, 13(5):1006–1029, 2009a.

- P. S. Oliveto, P. K. Lehre, and F. Neumann. Theoretical analysis of rank-based mutation-combining exploration and exploitation. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1455–1462, Trondheim, Norway, 2009b.
- P. S. Oliveto, T. Paixão, J. P. Heredia, D. Sudholt, and B. Trubenová. How to escape local optima in black box optimisation: When non-elitism outperforms elitism. *Algorithmica*, 80(5):1604–1633, 2018.
- T. Paixão, J. Pérez Heredia, D. Sudholt, and B. Trubenová. First steps towards a run-time comparison of natural and artificial evolution. In *Proceedings of the 17th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1455–1462, Madrid, Spain, 2015.
- I. Partalas, G. Tsoumakas, and I. Vlahavas. A study on greedy algorithms for ensemble pruning. Technical Report TR-LPIS-360-12, Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece, 2012.
- R. Poli, W. Langdon, and N. McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, Raleigh, NC, 2008.
- R. Poli, L. Vanneschi, W. Langdon, and N. McPhee. Theoretical results in genetic programming: The next ten years? *Genetic Programming and Evolvable Machines*, 11(3):285–320, 2010.
- A. Prügel-Bennett, J. Rowe, and J. Shapiro. Run-time analysis of population-based evolutionary algorithm in noisy environments. In *Proceedings of the 13th International Workshop on Foundations of Genetic Algorithms (FOGA)*, pages 69–75, Aberystwyth, UK, 2015.
- C. Qian, Y. Yu, and Z.-H. Zhou. On algorithm-dependent boundary case identification for problem classes. In *Proceedings of the 12th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 62–71, Taormina, Italy, 2012.
- C. Qian, Y. Yu, and Z.-H. Zhou. An analysis on recombination in multi-objective evolutionary optimization. *Artificial Intelligence*, 204:99–119, 2013.
- C. Qian, Y. Yu, and Z.-H. Zhou. Pareto ensemble pruning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI)*, pages 2935–2941, Austin, TX, 2015a.
- C. Qian, Y. Yu, and Z.-H. Zhou. On constrained Boolean Pareto optimization. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 389–395, Buenos Aires, Argentina, 2015b.
- C. Qian, Y. Yu, and Z.-H. Zhou. Subset selection by Pareto optimization. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28 (NIPS)*, pages 1765–1773. Curran Associates, Inc., Red Hook, NY, 2015c.
- C. Qian, Y. Yu, and Z.-H. Zhou. Variable solution structure can be helpful in evolutionary optimization. *Science China Information Sciences*, 58(11):1–17, 2015d.
- C. Qian, J.-C. Shi, Y. Yu, K. Tang, and Z.-H. Zhou. Parallel Pareto optimization for subset selection. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1939–1945, New York, NY, 2016a.
- C. Qian, Y. Yu, and Z.-H. Zhou. A lower bound analysis of population-based evolutionary algorithms for pseudo-Boolean functions. In *Proceedings of the 17th International Conference on Intelligent Data Engineering and Automated Learn-*

- ing (*IDEAL*), pages 457–467, Yangzhou, China, 2016b.
- C. Qian, J.-C. Shi, Y. Yu, K. Tang, and Z.-H. Zhou. Subset selection under noise. In I. Guyon, U. V. Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30 (NIPS)*, pages 3563–3573. Curran Associates, Inc., Red Hook, NY, 2017a.
- C. Qian, J.-C. Shi, Y. Yu, K. Tang, and Z.-H. Zhou. Optimizing ratio of monotone set functions. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2606–2612, Melbourne, Australia, 2017b.
- C. Qian, C. Bian, Y. Yu, K. Tang, and X. Yao. Analysis of noisy evolutionary optimization when sampling fails. In *Proceedings of the 20th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1507–1514, Kyoto, Japan, 2018a.
- C. Qian, J.-C. Shi, K. Tang, and Z.-H. Zhou. Constrained monotone k -submodular function maximization using multi-objective evolutionary algorithms with theoretical guarantee. *IEEE Transactions on Evolutionary Computation*, 22(4):595–608, 2018b.
- C. Qian, Y. Yu, K. Tang, Y. Jin, X. Yao, and Z.-H. Zhou. On the effectiveness of sampling for evolutionary optimization in noisy environments. *Evolutionary Computation*, 26(2):237–267, 2018c.
- C. Qian, Y. Yu, and Z.-H. Zhou. Analyzing evolutionary optimization in noisy environments. *Evolutionary Computation*, 26(1):1–41, 2018d.
- J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan kaufmann, San Francisco, CA, 1993.
- G. R. Raidl. An improved genetic algorithm for the multiconstrained 0-1 knapsackproblem. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 207–211, Anchorage, AK, 1998.
- R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 475–484, El Paso, TX, 1997.
- E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 2902–2911, Sydney, Australia, 2017.
- O. Reyes and S. Ventura. Evolutionary strategy to perform batch-mode active learning on multi-label data. *ACM Transactions on Intelligent Systems and Technology*, 9(4):46, 2018.
- J. Richter, A. Wright, and J. Paxton. Ignoble trails-where crossover is provably harmful. In *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 92–101, Dortmund, Germany, 2008.
- V. Rijsbergen and C. Joost. Foundation of evaluation. *Journal of Documentation*, 30(4):365–373, 1974.
- M. F. Rogers, A. Howe, and D. Whitley. Looking for shortcuts: Infeasible search analysis for oversubscribed scheduling problems. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 314–323, Cumbria, UK, 2006.

- L. Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39, 2010.
- J. S. Rosenthal. Minorization conditions and convergence rates for Markov chain Monte Carlo. *Journal of the American Statistical Association*, 90(430):558–566, 1995.
- F. Rothlauf. *Representations for Genetic and Evolutionary Algorithms*. Springer, Berlin, Germany, 2006.
- J. Rowe and D. Sudholt. The choice of the offspring population size in the $(1,\lambda)$ evolutionary algorithm. *Theoretical Computer Science*, 545:20–38, 2014.
- G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5(1):96–101, 1994.
- G. Rudolph. *Convergence Properties of Evolutionary Algorithms*. Verlag Dr. Kovač, Hamburg, Germany, 1997.
- G. Rudolph. On a multi-objective evolutionary algorithm and its convergence to the Pareto set. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 511–516, Anchorage, AK, 1998.
- G. Rudolph and A. Agapie. Convergence properties of some multi-objective evolutionary algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1010–1016, Piscataway, NJ, 2000.
- T. P. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, 2000.
- G. Sasaki and B. Hajek. The time complexity of maximum matching by simulated annealing. *Journal of the ACM*, 35(2):387–403, 1988.
- J. Scharnow, K. Tinnefeld, and I. Wegener. Fitness landscapes based on sorting and shortest paths problems. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 54–63, Granada, Spain, 2002.
- B. Selman, H. A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the 12th AAAI Conference on Artificial Intelligence (AAAI)*, pages 337–343, Seattle, WA, 1994.
- B. Settles. Active learning literature survey. Technical Report 1648, Department of Computer Sciences, University of Wisconsin-Madison, Wisconsin, WI, 2010.
- D. Sharma, A. Deshpande, and A. Kapoor. On greedy maximization of entropy. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 1330–1338, Lille, France, 2015.
- C. Shi, X. Kong, D. Fu, P. S. Yu, and B. Wu. Multi-label classification based on multi-objective optimization. *ACM Transactions on Intelligent Systems and Technology*, 5(2):35, 2014.
- J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.
- A. P. Singh, A. Guillory, and J. Bilmes. On bisubmodular maximization. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1055–1063, La Palma, Canary Islands, 2012.
- A. Singla, S. Tschiatschek, and A. Krause. Noisy submodular maximization via adaptive sampling with applications to crowdsourced image collection summarization. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI)*,

- pages 2037–2043, Phoenix, AZ, 2016.
- P. Slavík. A tight analysis of the greedy algorithm for set cover. *Journal of Algorithms*, 25(2):237–254, 1997.
- P. Stagge. Averaging efficiently in the presence of noise. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 188–197, Amsterdam, The Netherlands, 1998.
- T. Storch. On the choice of the parent population size. *Evolutionary Computation*, 16(4):557–578, 2008.
- D. Sudholt. Crossover is provably essential for the ising model on trees. In *Proceedings of the 7th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1161–1167, Washington, DC, 2005.
- D. Sudholt. A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 17(3):418–435, 2013.
- D. Sudholt. How crossover speeds up building block assembly in genetic algorithms. *Evolutionary Computation*, 25(2):237–274, 2017.
- D. Sudholt and C. Thyssen. A simple ant colony optimizer for stochastic shortest path problems. *Algorithmica*, 64(4):643–672, 2012.
- T. Sun and Z.-H. Zhou. Structural diversity for decision tree ensemble learning. *Frontiers of Computer Science*, 12(3):560–570, 2018.
- R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press, Cambridge, MA, 1998.
- J. Suzuki. A Markov chain analysis on simple genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 25(4):655–659, 1995.
- C. Tamom and J. Xiang. On the boosting pruning problem. In *Proceedings of the 11th European Conference on Machine Learning (ECML)*, pages 404–412, Barcelona, Spain, 2000.
- J. Thapper and S. Živný. The power of linear programming for valued CSPs. In *Proceedings of the 53rd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 669–678, New Brunswick, NJ, 2012.
- R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.
- J. A. Tropp. Greed is good: Algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242, 2004.
- S. Venkatraman and G. G. Yen. A generic framework for constrained optimization using genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 9(4):424–435, 2005.
- J. Vondrák. Submodularity and curvature: The optimal algorithm. *RIMS Kokyuroku Bessatsu B*, 23:253–266, 2010.
- M. Wagner and F. Neumann. Parsimony pressure versus multi-objective optimization for variable length representations. In *Proceedings of the 12th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 133–142, Taormina, Italy, 2012.
- M. Wagner and F. Neumann. Single-and multi-objective genetic programming: New runtime results for sorting. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 125–132, Beijing, China, 2014.

- Z. Wang, E. Chen, Q. Liu, Y. Yang, Y. Ge, and B. Chang. Maximizing the coverage of information propagation in social networks. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2104–2110, Buenos Aires, Argentina, 2015.
- J. Ward and S. Živný. Maximizing bisubmodular and k -submodular functions. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1468–1481, Portland, OR, 2014.
- R. A. Watson. Analysis of recombinative algorithms on a non-separable building-block problem. In *Proceedings of the 6th International Workshop on Foundations of Genetic Algorithms (FOGA)*, pages 69–89, Charlottesville, VA, 2001.
- I. Wegener. Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions. In R. A. Sarker, M. Mohammadian, and X. Yao, editors, *Evolutionary Optimization*, pages 349–369. Kluwer, Norwell, MA, 2002.
- C. Witt. Worst-case and average-case approximations by simple randomized search heuristics. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 44–56, Stuttgart, Germany, 2005.
- C. Witt. Runtime analysis of the $(\mu+1)$ EA on simple pseudo-Boolean functions. *Evolutionary Computation*, 14(1):65–86, 2006.
- C. Witt. Population size versus runtime of a simple evolutionary algorithm. *Theoretical Computer Science*, 403(1):104–120, 2008.
- C. Witt. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability and Computing*, 22(2):294–318, 2013.
- D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.
- L. A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.
- X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.
- Y. Yu and C. Qian. Running time analysis: Convergence-based analysis reduces to switch analysis. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 2603–2610, Sendai, Japan, 2015.
- Y. Yu and Z.-H. Zhou. A new approach to estimating the expected first hitting time of evolutionary algorithms. *Artificial Intelligence*, 172(15):1809–1832, 2008a.
- Y. Yu and Z.-H. Zhou. On the usefulness of infeasible solutions in evolutionary search: A theoretical study. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 835–840, Hong Kong, China, 2008b.
- Y. Yu, C. Qian, and Z.-H. Zhou. Towards analyzing recombination operators in evolutionary search. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 144–153, Krakow, Poland, 2010.
- Y. Yu, C. Qian, and Z.-H. Zhou. Towards analyzing crossover operators in evolutionary search via general Markov chain switching theorem. CORR abs/1111.0907, 2011.
- Y. Yu, X. Yao, and Z.-H. Zhou. On the approximation ability of evolutionary optimization with application to minimum set cover. *Artificial Intelligence*, 180-181: 20–33, 2012.

- Y. Yu, C. Qian, and Z.-H. Zhou. Switch analysis for running time analysis of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 19(6):777–792, 2015.
- C.-H. Zhang. Nearly unbiased variable selection under minimax concave penalty. *Annals of Statistics*, 38(2):894–942, 2010.
- T. Zhang. Adaptive forward-backward greedy algorithm for learning sparse representations. *IEEE Transactions on Information Theory*, 57(7):4689–4708, 2011.
- Y. Zhang, S. Burer, and W. N. Street. Ensemble pruning via semi-definite programming. *Journal of Machine Learning Research*, 7:1315–1338, 2006.
- H. Zhou. Matlab SparseReg Toolbox Version 0.0.1. 2013.
- H. Zhou, A. Armagan, and D. Dunson. Path following and empirical Bayes model selection for sparse regression. CORR abs/1201.3528, 2012.
- Y. Zhou and J. He. A runtime analysis of evolutionary algorithms for constrained optimization problems. *IEEE Transactions on Evolutionary Computation*, 11(5):608–619, 2007.
- Z.-H. Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC, Boca Raton, FL, 2012.
- Z.-H. Zhou. A brief introduction to weakly supervised learning. *National Science Review*, 5(1):44–53, 2017.
- Z.-H. Zhou, J. Wu, and W. Tang. Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 137(1):239–263, 2002.
- X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, Department of Computer Sciences, University of Wisconsin-Madison, Wisconsin, WI, 2008.