NOISY COMBINATORIAL OPTIMISATION WITH EVOLUTIONARY ALGORITHMS

by

AISHWARYAPRAJNA

A thesis submitted to the University of Birmingham

for the degree of

DOCTOR OF PHILOSOPHY

School of Computer Science

College of Engineering and Physical Sciences

University of Birmingham

December 2021

Abstract

The determination of the efficient evolutionary optimisation approaches in solving noisy combinatorial problems is the main focus in this research. Initially, we present an empirical study of a range of evolutionary algorithms applied to various noisy combinatorial optimisation problems. There are four sets of experiments. The first looks at several toy problems, such as OneMax and other linear problems. We find that Univariate Marginal Distribution Algorithm (UMDA) and the Paired-Crossover Evolutionary Algorithm (PCEA) are the only ones able to cope robustly with noise, within a reasonable fixed time budget. In the second stage, UMDA and PCEA are then tested on more complex noisy problems: SubsetSum, Knapsack and SetCover. Both perform well under increasing levels of noise, with UMDA being the better of the two. In the third stage, we consider two noisy multi-objective problems (CountingOnesCountingZeros and a multi-objective formulation of SetCover). We compare several adaptations of UMDA for multi-objective problems with the Simple Evolutionary Multi-objective Optimiser (SEMO) and NSGA–II. In the last stage of empirical analysis, a realistic problem of the path planning for the ground surveillance with Unmanned Aerial Vehicles is considered. We conclude that UMDA, and its variants, can be highly effective on a variety of noisy combinatorial optimisation, outperforming many other evolutionary algorithms.

Next, we study the use of voting mechanisms in populations, and introduce

a new Voting algorithm which can solve OneMax and Jump in $O(n \log n)$, even for gaps as large as $O(n)$. More significantly, the algorithm solves OneMax with added posterior noise in $O(n \log n)$, when the variance of the noise distribution is $\sigma^2 = O(n)$ and in $O(\sigma^2 \log n)$ when the noise variance is greater than this. We assume only that the noise distribution has finite mean and variance and (for the larger noise case) that it is unimodal. Building upon this promising performance, we consider other noise models prevalent in optimisation and learning and show that the Voting algorithm has efficient performance in solving OneMax in presence of these noise variants. We also examine the performance on arbitrary linear and monotonic functions. The Voting algorithm fails on LeadingOnes but we give a variant which can solve the problem in $O(n \log n)$. We empirically study the use of voting in population based algorithms (UMDA, PCEA and cGA) and show that this can be effective for large population sizes.

To Ma-Baba...

Acknowledgements

Firstly, I would like to express my deepest gratitude to my supervisor, Professor Jonathan E. Rowe for his erudite guidance, invaluable insights and encouragement through all times. His perceptive knowledge sharing and his relentless efforts and patience pushed me to sharpen my thinking and improve the research quality. His kind support, even during the pandemic, helped me immensely to complete this work. I thank my thesis group members deeply for their support and suggestions. I am indebted to School of Computer Science, University of Birmingham for providing me the funding and resources for this research. I sincerely thank Prof. Juergen Branke and Dr. Miqing Li and for their advice that helped in improving the thesis.

I am thankful to Prof. Kirubarajan and Dr. Tharmarasa from McMaster University, Canada for kindly providing me support and guidance during our research collaboration. I am grateful to U21-Birmingham Global for enabling this with the PhD Research scholarship.

This work would never have been possible without the support of my family and friends, I would like to thank all of them. I am immensely indebted for all the love and blessings from Guruji and Guruma. I thank Sanchita-Mashi and Mesho tremendously for making me feel at home in Birmingham.

I am not capable to express my acknowledgement to Ma and Baba for their love, patience, motivation and advice throughout. I don't know how to thank Abhijit enough, for being the most patient and loving partner during my PhD journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

" What is not surrounded by uncertainty cannot be the truth."

— ***Richard P. Feynman***

Uncertainties are inherent in realistic decision-making processes. The unwanted perturbations in any quantifiable system are mostly modelled with probabilistic approaches and are often defined with an uncountable noun *noise*. The word *noise* is typically used to represent a multitude of uncertainties in physical systems including, sensory measurement errors, randomized simulations, computational inaccuracies, approximations in analytical methods, and perturbations arising due to external factors. Making realistic decisions by optimisation requires careful consideration of the random noise mingled up with the system while weighing in the plausible options.

Since noise (or, randomness, or, uncertainty, in general) is highly prevalent in nature and living organisms have struggled for their survival in nature, according to the Darwinian theory of evolution, it may be said that their fight for existence has made them adaptive to noisy interferences. This idea has driven researchers to hypothesise that randomised heuristics like Genetic Algorithms (GAs), designed on the basis of Darwin's theory, may have robustness to noisy environments. A considerable amount of literature has been published in favour of Evolutionary

Algorithms (EAs) being an advantageous choice for optimisation problems (mostly, in continuous spaces) in the presence of noise [11].

Noisy fitness functions are often used to approximately assess a fitness function when the actual fitness function is not known or, when noisy fitness evaluations are much faster to evaluate than accurate fitness functions. Noisy fitness evaluations might arise from incorrect or partial information as well. However, noisy fitness evaluations (or objective function evaluations) deceive the EA in finding the optimal solution by overestimating the inferior and underestimating the superior candidate solution in many instances. Since the EAs' target is to find the fittest search string (solution), it is likely that the algorithm would slow down its convergence to the optimum or get stuck at a non-optimal solution in the presence of noise.

Optimisation in the presence of noise has received considerable attention in the case of continuous optimisation, and a large number of heuristic approaches have been explored [66, 97]. However, until recently, there were fewer studies for combinatorial problems in the context of noise [11]. As well as presenting interesting theoretical challenges, handling noise in combinatorial optimisation can be important in practice. For example, for a route-finding problem, uncertainties will occur due to varying and unknown traffic conditions. In classification and regression tasks, noisy data labels or imperfect and incomplete data sets are often encountered. Many engineering design problems utilise stochastic simulations in evaluating potential solutions. Sometimes, even the actual objective function is not known. It is important, therefore, to identify algorithms that are resilient in the scenario of black box combinatorial optimisation when noise and uncertainties are present in the optimisation environment.

This thesis aims to understand which heuristic algorithms or mechanisms, especially evolutionary, may be beneficial when a combinatorial optimisation problem is in hand, and realistic noisy interferences in the system are in consideration. The next sections discuss the major research questions that motivate the research in this thesis and the research contributions of the thesis.

1.1 Motivational Research Questions in Noisy Combinatorial Optimisation

There has been recent theoretical research on the performance of EAs on combinatorial problems [85]. However, in the context of noisy combinatorial optimisation, not much is known regarding the best choice of an EA. The following main questions motivated the research in this thesis.

- ***Which EA performs the best in solving simple problems?***

  Recent theoretical research on the runtime of EAs (i.e., fitness evaluations needed to reach optimum) has investigated the benchmark OneMax, i.e. the bit-counting problem, with added noise in fitness evaluations. Polynomial runtime bounds have been provided for several EAs — including mutation-based algorithms with and without a population, crossover-only algorithm and estimation of distribution algorithm (EDA). However, it is important to study whether any of these algorithms are efficient in practice, within a reasonable, but, fixed fitness function evaluation budget and when the variance of the added noise is large.

- ***Can the algorithm solve harder problems? How much noise can the algorithm tolerate?***

  It is a concern whether the algorithms that are capable of handling noise with high variance in the case of the simple problem OneMax, can efficiently solve harder combinatorial problems without and with constraints, like, the Linear problem, the SubsetSum problem, the Knapsack problem or, the SetCover problem with large additive noises within fixed runtime budgets.

- ***How to solve noisy multi-objective combinatorial problems?***

  In the context of noisy multi-objective optimisation, the challenge is that each time a fitness comparison is made, there is an added random noise in the fitness value. Although there has been significant research on noisy multi-objective optimisation in continuous domains, there has been little research when the problem is combinatorial in nature. An important aspect of this involves investigation of the efficient algorithms and relevant diversification mechanisms, that provide a well-spread set of Pareto optimal solutions.

- ***How to solve harder and real-world applications that need combinatorial multi-objective optimisation, but are noisy?***

  There is an interesting application domain for assessing the efficiency of the better performing multi-objective algorithms — a complex realistic problem of planning the surveillance path of a ground region with an Unmanned Aerial Vehicle (UAV), with resource constraints and uncertainties arising due to weather factors.

- ***Can we think beyond EAs for noisy combinatorial optimisation?***

  Building upon the prior knowledge from theory and our empirical analysis that a population and crossover (or crossover-like) mechanisms are helpful in noisy combinatorial optimisation, an important question is whether a voting algorithm employing majority-vote recombination, can be the most efficient in solving noisy OneMax theoretically as well as practically. Here, if the noise arises from several perspectives prevalent in optimisation and machine learning scenarios, it is interesting to study if the same algorithm would be beneficial. Also, if this algorithm works well on its own, it is interesting to investigate if it can offer speed-ups when combined with existing algorithms.

1.2 Thesis Outline & Publications

In this section, a brief discussion of the chapters are provided. Later, a list of publications (that are already published and are in process) from the research in this thesis is given.

The Chapter 2 starts with a discussion of the different models of noise that are generally studied or are relevant in the context of combinatorial optimisation with evolutionary algorithms or machine learning problems. The common noise handling techniques prevalent in empirical and theoretical analyses are analytically studied and a collection of EAs, including, (1+1)–EA, Mutation-Population Algorithm, Compact GA (cGA), Population-Based Incremental Learning (PBIL), Univariate Marginal Distribution Algorithm (UMDA) and Paired-Crossover Evolutionary Algorithm (PCEA), are chosen carefully to analyse their performances.

In Chapter 3, an experimental comparison of this collection of algorithms on noisy OneMax and noisy Linear problems with additive Gaussian noise is presented, to see which algorithms can find solutions within a reasonable amount of function evaluations, bearing in mind that the asymptotic bounds for some of these algorithms in existing literature, while being polynomial, are actually very large. It may be concluded that UMDA and PCEA are the only ones able to cope robustly with noise for OneMax, within a reasonable fixed time budget.

The UMDA and PCEA are then tested on more complex noisy problems: SubsetSum, Knapsack and SetCover (as a constrained problem and with a penalty function) in Chapter 4 to see how well they handle noise. These problems are chosen, as they have a 'packing' structure which might make them amenable to algorithms which can solve noisy OneMax efficiently.

In Chapter 5, noisy multi-objective problems are considered. Initially, an empirical analysis of the performance of a collection of multi-objective evolutionary algorithms (MOEAs) on a toy multi-objective problem CountingOnesCountingZeroes (COCZ) without and with high levels of noise is done to identify which algorithms perform better. The mutation-based, Simple Evolutionary Multi-Objective Optimiser (SEMO), the Non-dominated Sorting Genetic Algorithm–II (NSGA–II), which is widely studied, and some multi-objective versions of UMDA with various diversification mechanisms are studied. The algorithms are compared on the basis of the hypervolume performance indicator, which provides an analysis of the spread of the non-dominated solutions found, in a reasonable time budget. Then, the noisy constrained SetCover problem is formulated as a multi-objective problem and an empirical analysis is performed to identify whether the constrained, penalty function version or the multi-objective version is faster to solve.

In Chapter 6, a UAV path planning scenario in presence of weather factors occluding the visibility of the UAV, is formulated as a noisy combinatorial multi-objective problem. Solution methodologies of the problem with MOEAs adapted to the UAV problem are proposed. This work was done during a short-term research collaboration in Summer 2020 with Department of Electrical and Computer Engineering, McMaster University, Canada, funded by the Birmingham Global Universitas21 (U21) PhD scholarship.

Chapter 7 discusses the use of voting mechanisms in populations, and the introduction of a new Voting algorithm which can solve OneMax and Jump in $O(n \log n)$, even for gaps as large as $O(n)$. More significantly, the algorithm solves OneMax with added unimodal posterior noise having a finite mean and variance in $O(n \log n)$, when the variance of the noise distribution is $\sigma^2 = O(n)$ and in $O(\sigma^2 \log n)$ when the noise variance is greater than this. This is the best performance of an algorithm on noisy OneMax known to date. Analyses show that the voting algorithm is also efficient for several other noise models. The performance on arbitrary linear and monotonic functions are also investigated. In practice, the use of voting mechanism in several population-based EAs (UMDA, cGA and PCEA) show significant speed-ups when the population sizes are large.

Chapter 8 discusses the concluding remarks of the thesis and the future scopes of research.

Some of the research works from this thesis have been published and some are in the the process of publication, as follows,

1. Aishwaryaprajna and Rowe, J.E., 2019, July. Noisy combinatorial optimisation by evolutionary algorithms. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (pp. 139-140) [4].

2. Rowe, J.E. and Aishwaryaprajna, 2019, August. The benefits and limitations of voting mechanisms in evolutionary optimisation. In Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms (pp. 34-42). [100]

3. Aishwaryaprajna and Rowe, J.E., 2021. Evolutionary Algorithms for Solving Unconstrained, Constrained and Multi-objective Noisy Combinatorial Optimisation Problems. arXiv preprint arXiv:2110.02288 [3]. *(Submitted to Journal).*

4. Aishwaryaprajna and Rowe, J.E. The Voting algorithm is robust to various noise models. *(Submitted to Journal)*

5. Aishwaryaprajna, Kirubarajan, T., Rowe, J.E., and Tharmarasa, R. UAV Path Planning in Presence of Occlusions as Noisy Combinatorial Multiobjective Optimisation. *(Submitted to Journal)*

Chapter 2

State of the Art

Since early research on evolutionary computation [26], noisy optimisation has attracted significant attention of researchers. Firstly, this chapter discusses several variants of noise that arise in combinatorial optimisation or, in certain machine learning scenarios defined in discrete spaces. Secondly, the common noise handling strategies are discussed. The theoretical and empirical results from the literature (mostly, on simple combinatorial test problems) are categorised on the basis of structural differences in the evolutionary optimisation algorithms. This is followed by a short outline of the literature on constrained and multi-objective optimisation in the context of noisy combinatorial problems.

In recent years, researchers have started analysing the runtime of EAs for solving combinatorial optimisation problems [85]. The runtime analysis of EAs involves estimating the time (function evaluations) required to obtain the optimum of a problem. In this context, the optimisation of pseudo-Boolean test functions are often studied theoretically. One of the most widely analysed test functions defined on a binary search string of length n is the OneMax problem : $\{0,1\}^n \to \mathbb{N}$, where the number of ones are to be maximised.

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

More generally, the optimisation process tries to guess a hidden binary string using several trials with the help of the fitness oracle, that provides the Hamming distance of the random string with the hidden string. The OneMax function is chosen as a starting point for analysing harder and more complex problems due to its uniformity and its independence within the bits [78].

This chapter concludes with a summary of existing theoretical runtime bounds for the noisy OneMax problem. This summary provides an idea to identify which algorithms or evolutionary mechanisms may be beneficial in noisy combinatorial optimisation and the performance of the obtained collection of algorithms is analysed with experiments in the following chapter(s).

2.1 Uncertainties in Combinatorial Optimisation

In real-world problems, noise is pervasive and can arise from various factors, for example, in scheduling problems, uncertainties can stem from unknown order arrival and delivery times, machinery disorders and variable processing times. It is important to therefore identify algorithms that are resilient to various kinds of noise in combinatorial optimisation.

From the perspective of evolutionary computation while optimising combinatorial functions, uncertainties in optimisation in the form of randomness can stem from a range of factors including (but, not limited to) noisy fitness evaluations, search string being noisy or, even incorrect fitness comparisons. In certain instances, the fitness function may be complicated and computationally expensive, so an approximate/sampled fitness function is evaluated. This brings sampling noise into the optimisation system. In some cases, the problem parameters are random in nature and the fitness evaluation for the same search string might be different in each instance. Sometimes, the whole of the search string might not be accessed during the optimisation process, which would impact the fitness evaluation of the search string. These uncertainties can disrupt the natural course of the behaviour of an EA.

Several machine learning problems like classification or clustering are combinatorial in nature [60]. Noise is often pervasive in these problems of data mining. For example, a dataset containing the 0–1 labels of n examples might have incorrect labels arising in the data collection and processing stages. This kind of noise is very similar to having a noisy search point which has random bits flipped in an optimisation process. A range of several noise variants considering a multitude of uncertainties in combinatorial optimisation or machine learning problems alike, has been considered in this thesis.

2.1.1 Posterior Noise

The posterior noise comes into play when the fitness value of the search point under consideration, has presence of noise in it after evaluation of the fitness function. For a combinatorial problem with posterior noise, the fitness function at each evaluation receives an addition of a random value drawn from some probability distribution η with finite mean and variance σ^2.

$$f^{noisy}(x) = f(x) + \eta(\sigma^2)$$

This noise can affect the optimisation severely when the variance is large. In some recent theoretical studies, the runtime bounds have been estimated while optimising in presence of posterior noise [6, 22, 47, 91].

The presence of posterior noise in optimisation has been considered in several experiments with large noise variances in Chapters 3–7. A more complex version of this noise variant is studied in Chapter 4 for a constrained problem Knapsack, where the noise enters the constraint evaluation. A theoretical study of the OneMax problem in presence of posterior noise is carried out in Chapter 7.

2.1.2 Prior Noise

The prior noise arises when instead of evaluating the considered search point, a noisy search point is evaluated. The prior noise flips a single bit (also known as, one bit prior noise) or multiple bits (also known as, bitwise prior noise) in the search point before the fitness evaluation is performed. The generalised multiple bit-flipping prior noise, can be defined as follows,

$$f^{noisy}(x) = \begin{cases} f(x) & \text{with probability } (1-p) \\ f(x') & \text{with probability } p \end{cases}$$

where, x' is generated by independently flipping each bit of x with probability q. When the pseudo-Boolean problem is affected by the prior bit-flipping noise, a bit gets flipped with probability pq. Some recent theoretical analyses [35, 93, 71, 111] have considered this variant of noise in analysing the performance of EAs (summary in section 2.6). The OneMax problem in presence of prior noise is studied theoretically in Chapter 7 while optimising with the Voting algorithm.

2.1.3 Partial Evaluation of Fitness Functions

In noisy data mining and learning problems, incomplete or unavailable data attributes are often encountered. In optimisation problems, a similar noise model is often encountered where the fitness evaluation occurs on an incomplete search string. Here, a noise model with partial evaluation of the fitness function of a binary search string is considered. During the fitness evaluation, only some bits containing a one in the search string are considered in the fitness evaluation with a specific probability. The effect of this noise model on the problems, OneMax and LeadingOnes is studied for a non-elitist binary selection algorithm and (1+1)–EA in [23]. In Chapter 7, the runtime of the Voting algorithm is studied, on the

OneMax problem in the presence of the same noise model as studied in [23].

2.1.4 Sampled Fitness Functions

In several machine learning scenarios, the learner does not have access to all the data attributes. A similar variant of noise is also prevalent in optimisation problems, where only a randomly chosen subset of data attributes (bits) is considered for fitness evaluation. A similar noise variant has been studied earlier in the context of computationally intensive and complex fitness functions [43, 52], where a subset of the search string is evaluated to obtain an approximate fitness value. This noise model where the fitness evaluation occurs on a subset of bits is studied is presence of the OneMax problem with the Voting algorithm in Chapter 7.

2.1.5 Dynamic Functions

In a recent study [76] on the pseudo-Boolean Linear functions, the weights keep on changing in every generation, i.e., are dynamic in nature, however the optimum remains unchanged. This noise model would give completely different fitness values for the same search point in distinct generations. The effect of this noise model on the pseudo-Boolean BinVal function (problem definition given in Chapter 7) has been studied theoretically in Chapter 7.

2.1.6 Fitness Comparison Oracle Says Truth with a Probability

A comparison oracle takes two search strings as an input and provides the best string according to fitness as the output. These black-box oracles are often encountered in learning and optimisation problems. However, the decision provided by the comparison oracle may be noisy, i.e., the oracle says the truth with a specific probability. A recent study discusses the effect of uncertainty arising from noisy comparison oracle on the performance of clustering mechanisms [2]. In the selection step of EAs, in case of tournament selection, a noisy comparison oracle provides a wrong decision regarding the best string with a probability. The performance of the Voting algorithm in solving OneMax in presence of noisy comparison oracle is studied in Chapter 7 along with the formal definition of this noise variant.

2.1.7 Chance-Constrained Optimisation Problems

For optimising complex combinatorial problems, uncertainty may be associated with the constraints. In very recent research, there has been attention towards chance-constrained evolutionary optimisation for combinatorial problems where, the constraints have a thresholded probability of violation associated with them [83, 121, 29]. This category of uncertainty is often encountered in process optimisation and control.

Although, chance-constrained evolutionary optimisation is beyond the scope of our research work, it is relevant in this section as it is considered in the very few research papers on constrained combinatorial optimisation with evolutionary algorithms in presence of uncertainty. A discussion on these results is given in Section 2.4.

2.2 Noise Handling Strategies

Early research on EAs for noisy optimisation was mostly empirical, depending on theoretical analysis on the correlation of EA parameters. Sampling a fitness function several times to estimate its actual fitness is one of the most common noise handling strategies. The investigation of an appropriate size of samples and its correlation with population size along with other EA parameters have attracted researchers for a long time. Later, a class of algorithms was proposed, known as Estimation of Distribution Algorithms (EDAs), replaced the crossover and mutation mechanisms by sampling and updating a probability vector from the existing populations to generate newer populations. Some studies have investigated EDAs as well, in the context of noisy combinatorial optimisation. The following subsections discuss these noise handling strategies.

2.2.1 Population Sizing

One of the earliest questions that EA researchers looked at for the optimal algorithm performance/convergence, was how to choose appropriate population sizes. If the population is too small, then high-quality solutions may not be achieved. However, if the population is too large then, the computational resources will be overused unnecessarily. In presence of noise in the optimisation system, this question becomes even more difficult.

Perhaps, De Jong's thesis [26] was one of the first studies to look into the perspective of an optimal population size dependent on the noise and signal characteristics of the decision making problem. In one of the first studies to provide an algorithm convergence model, the empirical work by Fitzpatrick & Grefenstette in 1988 [43] considered a sampling fitness function instead of the accurate one to improve GA performance for a realistic noisy problem of image registration. The idea behind the sampling fitness function is that, the evaluation of a noisy fitness function may require fewer computational resources in many engineering problems, than the accurate fitness function. They developed a GA runtime model,

$$T = (\alpha + \beta s)GN$$

where, T is the total time required by a GA, G is the total number of generations, s is the sample size of the sampling fitness function and N is the population size. The constant α is the overhead time cost for an individual in a generation except for the fitness evaluation step and β is the cost for a single fitness evaluation (sample). It is assumed that if the sample is large enough, i.e., as large as the size of the search string, then the variance of noise in the sample becomes zero. According to their empirical results, the best performance of GA could be achieved within a limited time budget, with a small sample size but a large population size. However, it is a question if this notion of *implicit averaging* in terms of choosing a larger population size is true in general cases.

A study by Rattray & Shapiro [98] shows that when the population is made larger, the effects of noise on selection may be reduced, in the limits of weak selection. They have used Boltzmann selection, that has a dynamic selection pressure, which may be beneficial to model when Gaussian noise is present in the fitness evaluations.

The work by Miller & Goldberg [79] focused on understanding the optimal level of sampling by adding Gaussian noise to the fitness evaluations. Population sizing models for generalised problems (domains) and OneMax were used. This work assumes that the variance of the noise is related to sample size but is not zero, even when the sample is large. Based on the population sizing models in [43, 55], where the latter investigated population size with respect to fitness variance (depending on noise), the population size is given as,

$$N = \Gamma(\sigma_F^2 + \sigma_N^2) = \Gamma\left(\sigma_F^2 + \frac{\sigma_E^2}{s}\right)$$

where, Γ is a population sizing coefficient dependent on the problem/domain characteristics and the confidence level that the population will not prematurely converge. Here, σ_F^2 is the fitness variance of the initial population, σ_N^2 is the variance of noise in the noisy fitness function and σ_E^2 is the sampling fitness variance. Assuming the overhead computational costs due to sampling, the lower bound of optimal sampling size is shown as,

$$s* = \sqrt{\frac{\alpha}{\beta}\frac{\sigma_E^2}{\sigma_F^2}}$$

and justified further with empirical results on the OneMax. A similar result (given below), is documented in [78] that uses a less conservative population sizing model alike to [55, 57] (based on the Gambler's Ruin problem, an example of random walks) for obtaining a problem independent upper bound for OneMax with additive Gaussian noise.

$$N = \Gamma' \sqrt{\sigma_F^2 + \sigma_N^2}$$

Here, the population sizing coefficient Γ', is dependent only on the problem parameters.

In order to bring into consideration the noise induced from sampling since it approximates the actual fitness values instead of adding a random noise to fitness, the sampled OneMax problem (that evaluates OneMax fitness function by sampling bits in the binary search string) is studied empirically by Giguere & Goldberg [52] for an array of population sizing models [55, 78, 57, 52]. This work concluded that when the Gambler's Ruin population sizing model in [55, 78, 57] is used, sampling will not be much effective. Their study indicated that increasing the function evaluation budget is most beneficial in GA. However, this is inconvenient in realistic scenarios due to resource limitations if the convergence time of the algorithm is not under consideration.

In Chapter 7 of this thesis, a similar approximation in fitness evaluation with sampling is considered for the OneMax problem. A convergence result, in terms of the runtime of the algorithm (calculated by the function evaluations required to reach the optimum) of our newly introduced Voting Algorithm (described in Chapter 7) is discussed.

2.2.2 Adaptive Sampling

The key problem of determining the appropriate population size or sample size is that, in actual practice it requires problem-specific parameters and knowledge of the noise encountered. The work of Aizawa & Wah [5] focused on this difficulty of fixing the sample size beforehand and introduced the concept of adapting the number of samples of observed fitness of candidates in population, according to the different individuals in a generation. They studied two parameter-sizing scenarios on the performance on GA in presence of noise – the determination of a suitable duration of a generation and an appropriate sample size for each generation. The population size was assumed to be constant. They proposed assigning larger sample size when the generations increase and when the estimated variance of the individuals is large. However, the empirical studies in this paper are not performed on combinatorial functions.

The adaptive sampling method by Branke & Schmidt [13] took into account the error probability arising from noisy fitness function evaluations in the process of selection. They consider the stochastic tournament selection, where the better candidate is selected with a probability (modified as an error function). The method considers an adaptive sampling size that depends on the expected error probability in selection arising from the fitness difference of the individuals. They show that when individuals have similar fitness, the probability of having error in selection is highest. So it is proposed that the individuals with similar fitness values are to be sampled more often than the individuals which have different fitness values. Initially, a reduced sample size is chosen for both of the individuals and the sample size is increased gradually until a fixed threshold is reached. Then the observed fitness difference between two individuals with equal sized samples is checked, to decide which individual to choose. According to the empirical results, this method decreases the sample size significantly as the generations of the EA progresses (which will be helpful in reducing computation time). However, the convergence results and the final solution quality are not known in context of a noisy optimisation problem.

A subsequent work by Branke & Schmidt [14] studied sequential sampling techniques in order to reduce the selection error present in stochastic tournament selection for noisy problems for Gaussian noise. They propose a new sequential sampling technique, constant-error procedure and compare with the state-of-the-art Indifference Zone Selection procedure. In both of the methods the samples are generated until the observed fitness difference does not go beyond a threshold and the methods vary on the basis of choice of threshold and comparisons. Their method uses binary search to determine error probability and performs better in reducing the number of samples. Empirically, they studied OneMax problem of search strings with size n =1000 in presence of a very small Gaussian noise of mean 0 and standard deviation, σ = 0, 2, 5, 10. The results show that for $\sigma = 10$ and 80 samples per tournament, which means 40 samples per individual would need almost 4×10^6 function evaluations to obtain an average fitness of population of 800. However, it is not known if the optimum is obtained in the population.

Another study by Cantú-Paz [20] considers samples of the individual participating in the tournament with highest observed variance and selects the winner by using statistical tests. They considered 100-bit OneMaxproblem with noise strengths of σ = 10, 20 and demonstrated that adaptive sampling is beneficial than an excessively large fixed sample. However, it is evident from their results that, as the generations progress, since the individuals tend to become seemingly equal in presence of deceptive role of noise, the runtime becomes extremely large (of the order of 10^6) for a problem size of 100.

Therefore, the main question in adaptive and sequential sampling lies in whether these methods are effective in practice for finding the optimum for large levels of noise within a reasonable function evaluation budget in the context of combinatorial optimisation.

2.2.3 Estimation of Fitness Based on Neighbourbood

The idea of estimating the noisy fitness of an individual based on the fitness landscape of the neighbourhood was used in [12, 15, 102, 101]. It is assumed that the noise characteristics in the neighbourhood are similar and the fitness landscape is smooth locally. However, the experimentations in these studies are based on continuous spaces. It would be interesting to understand if this idea helps in noisy combinatorial problems, however, this investigation is not a part of this thesis.

2.2.4 Estimation of Distribution Algorithms

The EDAs sample an explicit probabilistic model derived from the most promising search strings found so far in the population, instead of using implicit probabilistic models as done by traditional EAs. The algorithms start from an initial population generated from a uniformly distributed probability vector. Then, the best solutions are chosen with respect to the fitness functions and the probability vector is updated. This updated probability vector generates the new population of search strings.

The compact GA (cGA) [58] is a simple EDA that considers a probability vector to represent the distribution of search strings in its implicit population (algorithm structure in Chapter 3). A paper by Sastry et al. [103] considers the noisy OneMax on instances with a large number of variables and compares the performance of mutation-based hillclimbers: sequential hillclimber and random hillclimber with a parallelized cGA. In this study, the considered Gaussian noise has a variance of 10^{-5} to 16 times the fitness variance. The paper concludes that the cGA is capable to solve problem with over 33 million variables. If the convergence criteria is relaxed such that when each bit in the probability vector reaches 0.501 it is declared as successful, then cGA can work with 1.1 billion variables. Parallelized versions of EDAs need to be investigated further for complex optimisations problems.

The performance of another EDA, univariate marginal distribution algorithm (UMDA) with binary tournament selection (algorithm structure in Chapter 3) was studied for the OneMax and the noisy OneMax problem by Pelikan & Sastry [87]. Using the gambler's ruin population sizing and some approximations, they show theoretically that the upper bound of the number of generations G needed for UMDA with binary tournament selection, to solve noisy OneMax is given by,

$$G = \frac{\pi}{2}\sqrt{\pi n}\sqrt{1 + \frac{\sigma_n^2}{\sigma_F^2}\left(1 - \frac{2\arcsin\left(2p_{init} - 1\right)}{\pi}\right)}$$

where, p_{init} is the probability of having a one in the initial population. This result is validated with experiments. They have considered p_{init} to study the effect of initial-population biasing towards the optimum. It is observed that when $p_{init} > 0.5$, i.e. the initial population has more ones, or the population is biased towards the optimum, the performance of UMDA improves while biasing the initial population away from the global optimum makes the performance worse. For empirical validation, the problem sizes $n = 100$ to 500 are considered.

The search strategy of UMDA includes only selection and recombination, since it is closely associated with the *genepool crossover*, where learning and sampling the probability vector corresponds to repeated crossover applications [82]. In order to understand the effects of the mutation operator in UMDA, they consider an interesting experiment of adding bit-flip mutation. They consider flipping each bit in a new search string with a bit-flipping probability.

The results show that UMDA with bit-flipping performs significantly worse (in terms of runtime and stability) than UMDA on non-noisy and noisy OneMax. However, the effects of initial-population bias in UMDA with mutation are not much significant. They conclude that any genetic algorithm that is mainly based on selection and recombination may have similar effects of initial-population biasing. These results indicate that plain UMDA has a certain resilience to noise and it requires thorough further investigation (experiments performed in later chapters). Also, theoretical analyses on population biasing is considered in Chapter 7 in the context of Voting algorithm.

2.3 Theoretical Runtime Analyses on Simple Noisy Combinatorial Problems

In recent years, there have been significant advances in the theoretical aspects on analysing EA performances on combinatorial optimisation problems [85, 62]. The next subsections categorise the theoretical results based on structural differences in the algorithms. This will provide a basis for identifying the relevant algorithms in performance analysis of EAs for noisy combinatorial optimisation.

2.3.1 Resampling in EAs

Theoretical runtime analyses of resampling for noisy optimisation has only been started in the recent years. In one of the most generalized results on resampling, Akimoto et al. [6] considered optimisation problems with additive noise in two configurations, viz., Gaussian and heavy tailed noise. They concluded that if the runtime of an algorithm is known in advance, then the algorithm may be modified by adding a fixed number of re-evaluations in each iteration for a noisy environment. They show that for a problem with additive Gaussian noise, the runtime increases from T (in the noise-free case) to $O(\sigma^2 T \log T)$, where σ^2 is the variance of the noise, this means that $\sigma^2 T \log T$ samples are required for each search point.

For example, we know that the expected runtime of (1+1)–EA solving the non-noisy OneMax is $T = O(n \log n)$. Then according to the result of Akimoto et al.

[6], the expected runtime of (1+1)–EA with resampling for solving the OneMax with additive Gaussian noise with variance σ^2 will have the following runtime:

$$\begin{aligned} O(\sigma^2 T \log T) &= O\left(\sigma^2 n \log n \log\left(n \log n\right)\right) \\ &= O\left(\sigma^2 n \log n (\log n + \log\log n)\right) \\ &= O\left(\sigma^2 n (\log n)^2 + \sigma^2 n \log n \log\log n\right) \\ &= O(\sigma^2 n (\log n)^2) \end{aligned}$$

However, it is important to bear in mind that knowledge regarding the variance of the noise is needed for using the result by Akimoto et al. in order to determine the number of samples that are needed to be considered. In realistic scenarios, accurate estimation of noise variance is particularly challenging. Also, obtaining accurate runtime in some noise-free scenarios may be particularly challenging.

For the domain of generalized OneMax of size n, the expected number of samples of the target function required by optimal elimination of fitness functions (OEFF) to eliminate all incompatible functions is $\Theta(\frac{n}{\log n})$, according to Anil & Wiegand [7]. In other words, the most effective algorithm to solve the non-noisy OneMax problem will have runtime of $T = \Theta(\frac{n}{\log n})$. By considering the results of Akimoto et al. [6] the most efficient algorithm using resampling to solve the noisy OneMax problem will have a runtime of $\Theta(\sigma^2 n)$.

Doerr and Sutton [32] proposed that median provides a better estimate than mean while resampling noisy fitness functions. In the analysis the noise is considered to be ε-concentrated, which is often the case with several realistic noise models including posterior additive Gaussian noise. However, we argue that this proposition holds only when σ is very small, in case of additive Gaussian noise of the form $N(0, \sigma^2)$. The argument is discussed as follows:

A random variable X is said to be ε-concentrated, (where, $0 < \varepsilon < 1/2$) when the following conditions are true simultaneously: (1) $\Pr\left(X \geq \frac{1}{2}\right) \leq \frac{1}{2} - \varepsilon$, (2) $\Pr\left(X \leq -\frac{1}{2}\right) \leq \frac{1}{2} - \varepsilon$. Since, the noise is symmetric, we can write the following from the definition of Gauss error function,

$$\Pr\left(X \geq \frac{1}{2}\right) = \Pr\left(X \leq -\frac{1}{2}\right) = \frac{1}{2} - \operatorname{erf}\left(\frac{1}{2\sigma}\right)$$

From the definition of ε-concentration, $\varepsilon = \operatorname{erf}\left(\frac{1}{2\sigma}\right)$. Since, $0 < \varepsilon < \frac{1}{2}$, from the definition of error function,

$$0 < \frac{1}{\sqrt{2\pi}} \int_0^{\frac{1}{2\sigma}} \exp^{-y^2/2} dy < \frac{1}{2}$$

$$\Rightarrow 0 < \left(\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{1}{2\sigma}} \exp^{-y^2/2} dy - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{0} \exp^{-y^2/2} dy \right) < \frac{1}{2}$$

$$\Rightarrow 0 < \left(\Phi\left(\frac{1}{2\sigma}\right) - \frac{1}{2} \right) < \frac{1}{2}$$

Then we can conclude that only when the standard deviation of the noise is very small, i.e., $0 < \sigma < 0.9538$, the proposition that median might be a more beneficial choice than mean holds true. Also, according to the theorem 3.1 in [32], the runtime of the median-resampling version of the randomised search heuristic is $O(T\varepsilon^{-2} \log T)$ where T is the runtime for the non-noisy function. Here, if the following Taylor series expansion is considered,

$$\frac{1}{\varepsilon^2} = \frac{1}{\operatorname{erf}(\frac{1}{2\sigma})^2} = \Theta(\sigma^2)$$

then the runtime bound for the algorithm with median-resampling [32] is similar to the one having a mean-resampling [6] strategy.

It is also to be noted that the voting algorithm [100] (refer Chapter 7 for details) solves OneMax with a unimodal noise distribution with finite mean and variance (for the larger noise case) or the Paired-Crossover Evolutionary Algorithm (PCEA) [91] solves OneMax for additive Gaussian noise with a better runtime than resampling. Consequently, resampling has not been considered in our study of noisy combinatorial optimisation problems.

2.3.2 Mutation-based Algorithms

The well known hill-climbing algorithm (1+1)–EA chooses an arbitrary candidate solution and attempts to find a better solution by flipping (mutating) each bit in the search string with a probability, until no further improvements can be achieved (detailed algorithm structure in Chapter 3).

Droste [35] performed the first runtime analysis of EAs on noisy functions by analysing the performance of (1+1)–EA for the OneMax problem with one-bit prior noise that flips a single bit with probability p prior to the function evaluation. The paper concludes that (1+1)–EA is capable to optimize OneMax with high probability in polynomial time if and only if p is $O(\log n/n)$.

The work by Droste was extended in [51] and generalised results were reproved with drift analysis [59, 73] for the one bit prior noise model. The runtime bounds of population-based mutation algorithms on the OneMax, $(\mu + 1)$-EA (having parent population) and $(1 + \lambda)$-EA (having offspring population) are given as $O\left(\frac{1}{p}n\log^2 n\right)$ and $O\left(\frac{1}{p}n^3\log n\right)$ when $p \in (0, 1]$, respectively. These bounds for the mutation-based algorithm with a population are slightly better than the plain iterative (1+1)–EA, with no population. Since population size can affect the ability of an EA to handle noise [55, 98] and the large pool of search strings may contain useful solutions for future generations, it is possible, therefore that the population can help the mutation algorithms to be efficient in noisy optimisation.

A recent study [72] has analysed the OneMax problem in presence of one bit and multiple bit prior noise models with a non-elitist population-based EA which mutates the tournament winner and adds it to the next population (refer to Chapter 3 for the detailed Mutation-Population algorithm description). The runtime bounds are given as are $O(n^2)$ and $o(n^3 \log n)$ for one bit and bit wise prior noise respectively. However, in realistic scenarios, the variance of one bit prior noise is very small. Although polynomial, the bounds for the bit wise prior noise are very large.

Several recent studies have analysed the LeadingOnes problem in runtime analysis. The LeadingOnes problem is another pseudo-Boolean function, that counts the maximum length of substring which has ones in all of its bits when the substring starts from the first bit position. It is defined as,

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

The optimisation of this problem requires step by step improvement in a particular sequence to reach the optimum. The threshold between the polynomial and super-polynomial expected runtime for the (1+1)–EA on LeadingOnes problem with one-bit prior noise is located at $p = \Theta(\log n/n^2)$ in [111].

The bit-wise noise characterised with parameters (p, q) for (1+1)–EA on OneMax and LeadingOnes are studied in [93]. The noise occurs in the fitness evaluation with a probability p and independently flips each bit of a solution with probability q before evaluation. They proved that the runtime is polynomial for the OneMax and the LeadingOnes problem with bitwise noise $(p, \frac{1}{n})$ if $p = O(\log n/n)$ and $p = O(\log n/n^2)$ respectively. They have shown that the expected running time of the algorithm is super-polynomial for the problems under the same bitwise noise, if $p = \omega(\log n/n) \cap 1 - \omega(\log n/n)$ and $p = \omega(\log n/n) \cap o(1)$ respectively. Whereas, the running time is exponential if $p = 1 - O(\log n/n)$ and $p = \Omega(1)$ respectively. Similar results are discussed for the bitwise noise $(1, q)$ and one-bit noise. These studies strongly indicate that including a population in the strategy of (1+1)–EA can slightly improve the performance, although the noise levels are still small.

Furthermore, Dang & Lehre studied the behavior of non-elitist EAs for OneMax and LeadingOnes having noise distributed as uniform, Gaussian and exponential. They considered a recently developed fitness-level theorem for non-elitist populations to estimate the expected running time for the said problems in a noisy environment. In case of additive Gaussian noise $N(0, \sigma^2)$ with mutation rate $\frac{\chi}{n} = \frac{a}{3\sigma n}$ and population size $\lambda = b\sigma^2 \ln n$, the considered algorithm optimizes the OneMax and the LeadingOnes problem in expected time $O(\sigma^7 n \ln(n) \ln(\ln(n)))$ and $O(\sigma^7 n \ln(n) + \sigma^6 n^2)$ respectively. Similar results were shown for uniform and exponential noise distributions. It must be noted that the variance of the noise needs to be known in advance.

The $(\mu+1)$-EA is a simple mutation-only algorithm, which randomly initialises a population of μ solutions and chooses a parent uniformly at random from that population. Friedrich et al. [49] showed that the $(\mu + 1)$ EA while solving the OneMax cannot cope with the additive Gaussian noise with variance σ^2, regardless of the population size.

However, Friedrich et al. in [48] considered OneMax problem with an additive posterior noise arising from some random probability distribution D and investigated which properties of D could lead to a graceful scaling by the $(\mu+1)$-EA. The paper introduced a concept *'graceful scaling'* in which the runtime of an algorithm scales polynomially with noise intensity. When the case of exponentially decaying tails in the distribution of the noise is considered, which is quite similar to the noise which is Gaussian, it is shown that $(\mu + 1)$-EA is not capable to scale gracefully with noise. Whereas, in case of noise taken from the uniform distribution, the $(\mu + 1)$-EA gracefully scales with noise.

This was generalised to the $(\mu + \lambda)$–EA by Gießen and Kötzing [51], showing that populations can help in both prior and posterior noise. They show the (1+1)–

EA, however, can only tolerate posterior Gaussian noise when the variance is very small (less than $1/(4 \log n)$).

In any case, these bounds indicate that (1+1)–EA requires superpolynomial time to solve even small levels of posterior Gaussian noise, making plain (1+1)–EA an unreasonable choice for realistic noisy optimisation with significant noise. As a consequence, it seems likely that choosing local search for realistic noisy combinatorial optimisation will not be helpful.

2.3.3 Crossover-based Algorithms

Since early research into evolutionary algorithms, there has been a long debate on whether mutation or crossover is more beneficial. In 1991, the seminal paper by Goldberg et al. [55] considered performing enough recombinations with the help of a large population to obtain superior performance in noisy problems instead of a GA with small population size. In 1999, after a few papers proved that hill climbers are more efficient than genetic algorithms in some specific settings, in one of the earliest theoretical research papers on benefits of crossover, Jansen and Wegener [63] prove that genetic algorithms with mutation and uniform crossover can solve $\textsc{Jump}_{m,n}$ with polynomial runtime whereas, without crossover, the algorithms require superpolynomial time of the order $n^{\log n}$. The crossover probability considered is small, viz., $\frac{1}{n \log^3 n}$. They have investigated several variants of the genetic algorithm including one, which does not allow duplicates in the population. Later in 2005, they show that for Real Royal Road functions, mutation based evolutionary strategies require exponential time whereas, steady-state GAs can optimise in polynomial time [64].

Subsequently, researchers [112, 30, 21] have analysed that adding crossover to algorithms improves the algorithm efficiency. As a first study to understand the performance of crossover as a search operator in noisy combinatorial problems, Prügel-Bennett et al. [91] considered uniform crossover and selection only algorithm, known as Paired Crossover EA (PCEA) for the noisy OneMax problem. The crossover-based PCEA is capable to solve OneMax with noise of standard deviation $\sqrt{n}$ and population size $O(\sqrt{n}\log n)$ in expected time $O(n\log^2 n)$.

This study highlights an important insight that the larger step sizes taken by crossover are more beneficial than the small steps taken by hill climbers in presence of large noise. Detailed empirical studies on the performance of PCEA on simple and harder noisy combinatorial problems are shown in Chapters 3 and 4.

2.3.4 Estimation of Distribution Algorithms (EDAs)

A simple EDA, the compact GA (cGA) compares two individuals generated from a probability vector and an update of $\frac{1}{K}$ is made towards the winner of the tournament selection, where K is the population size. A detailed explanation of the cGA is given in Chapter 3 (please refer Algorithm 3). In the first rigorous runtime analysis of EDAs, Droste [34] shows that the expected runtime of cGA for any function is $\Omega(K\sqrt{n})$ and for any non-noisy linear function, it has an upper bound of $O(Kn)$.

Later, Friedrich et al. [49] show that cGA is capable to scale noise gracefully whereas the mutation only algorithm, (μ+1)-EA, fails to do so. They show that cGA is able to find the optimum of the OneMax problem with Gaussian noise of variance σ^2 after $O(K\sigma^2\sqrt{n}\log Kn)$ steps when $K = \omega(\sigma^2\sqrt{n}\log n)$, with probability $1 - o(1)$ [47]. However, please note that the number of steps required by the cGA is much higher than expected runtime of the Voting algorithm [100] (in Chapter 7) or PCEA [91] to solve the OneMax with the same levels of noise. This is illustrated with the help of empirical comparisons provided in Chapter 3.

A recent paper by Lehre & Nguyen [70] on the runtime analysis of UMDA has considered a one-bit prior noise model on the LeadingOnes function. The main result indicates that $O(n\lambda \log \lambda + n^2)$ function evaluations are needed considering optimal offspring population size $\lambda = O(n/\log n)$. However, research is needed for the runtime analysis of UMDA on other combinatorial optimisation problems for other noise models with large noise variances.

2.4 Noisy Combinatorial Optimisation with Constraints

In literature, several constraint handling techniques have been adopted in nature-inspired metaheuristics including EAs and swarm intelligence algorithms [77]. The techniques include representation of the constraints as a penalty function or, mapping the feasible region to a convenient search space, in order to avoid pitfalls in evolutionary optimisation like premature convergence and high computational costs.

It has been observed in multiple scenarios [99, 122] that multi-objective formulations of the constrained combinatorial problems enable to solve the problems in a significantly higher efficiency. Runtime analyses of constrained combinatorial problems have shown that adding constraints as another objective significantly improves the performance in problems like finding the minimum spanning trees and minimum vertex covers [84, 45].

Most of the recent studies on constrained combinatorial optimisation in presence of uncertainties have considered dynamic and chance-constrained [90, 108, 99, 83, 121, 29] problems. However, very little is known on the performance and runtime of EAs on combinatorial problems with constraints in presence of external noise variants. This thesis attempts to focus on this lacunae and has empirical studies with large values of posterior additive noise in complex and realistic constrained combinatorial problems (in Chapter 4).

2.5 Noisy Multi-Objective Combinatorial Optimisation

The Multi-Objective EAs (MOEAs) are capable to solve complex realistic problems. In order to make the MOEAs work, it needs to be ensured that they are able to obtain satisficing solutions that are as close as possible to the Pareto optimal set as well as diversity is maintained in the solution set. In order to assess the quality of successful approximations of the true Pareto optimal set, performance indicators like hypervolume comparison operator [125] are used.

In presence of noise, the dominance relationships in multi-objective optimisation change adversely for example, noise can make a strongly dominated solution as a member of a non-dominated solution set. This is particularly problematic for algorithms that make use of an *archive* of non-dominated solutions, as it is easy for a solution to be incorrectly placed in the archive due to the noise. However, in the context of noise, much of the previous work on multi-objective optimisation has concerned continuous problems ([54, 109, 42, 40]). In order to handle noise, techniques like probabilistic dominance and probabilistic Pareto ranking schemes [61, 113], resampling methods [19, 39, 86], modified elite preservation strategies [18] have been used. This thesis focuses on multi-objective combinatorial problems with additive (posterior) Gaussian noise and a detailed study with several diversification methods has been attempted (in Chapter 5).

2.6 Summary of Theoretical Bounds for Noisy Combinatorial Problems

In this section, comparisons of the theoretical runtime results of the well-studied test problem, OneMax problem with the popular noise models, the prior and posterior Gaussian noise are illustrated. These noise models and their parameters are discussed in a detailed manner in Chapter 7.

Algorithm	**Parameters**	**Runtime**	**n = 100** **p = 1**
(1+1)–EA	$\varrho = 1/n$, $p = O(1/n)$	$\Theta(n \log n)$ [51]	
	$\varrho = 1/n$, $p = O(\log n/n)$	polynomial [51]	
	$\varrho = 1/n$, $p = \omega(\log n/n)$	superpolynomial [51]	superpolynomial
(1+1)–EA with Sampling	$\varrho = 1/n$, Sampling size $k = 2$, $p = 1$	exponential[94]	exponential
	$\varrho = 1/n$, Sampling size $k = 4n^3$, $p = 1$	$O(n^4)$ [92]	~ 100000000
$(\mu + 1)$-EA	$\varrho = 1/n$, $\mu > 12\log(15n)/p$, $p \in (0, 1]$	$O(\mu n \log n)$ [51]	~ 2121
$(1 + \lambda)$-EA	$\varrho = 1/n$, $\lambda = \max\{12/p, 24\}$ $n \log n$, $p \in (0, 1]$	$O(n^2\lambda)$ [51]	~ 4605170
Non-elitist EA with tournament selection	$\varrho = 1/n$, $\lambda = O(\log n)$, $p \in [0, 1]$	$O(n \log n)$ [72]	~ 461

Table 2.1: Noisy OneMax with one bit prior noise. The calculations in the last column assume hidden constants = 1, for illustration purposes.

The Tables 2.1, 2.2 and 2.3 discuss the existing theoretical upper bounds which provide us an understanding regarding the algorithm performance in practice. In these comparisons, the runtime of our newly introduced Voting Algorithm (Chapter

7) is also included. All the expected runtimes in the these tables are exclusive of constant terms for illustration purposes. The mutation rate is denoted by ϱ in these tables.

Algorithm	**Parameters**	**Runtime**
(1+1)–EA	$\varrho = 1/n$, $p = O(1/n^2)$, $q = 1$	$\Theta(n \log n)$ [51]
	$\varrho = 1/n$, $p = O(\log n/n^2)$, $q = 1$	polynomial [51]
	$\varrho = 1/n$, $p = \omega(\log n/n^2)$, $q = 1$	$2^{\omega(\log n)}$ [51]
(1+1)–EA with Sampling	$\varrho = 1/n$, Sampling size $= O(n^3 + 2c)$, $p = \frac{1}{2} - \frac{1}{n^c}$, $c = \Phi(1)$, $q = 1$	polynomial [92]
Non-elitist EA with tournament selection	$\varrho = O(\log(1/n))$, $\lambda = \Omega\left(n \log n\right)$, $p \in (0, 1/2)$, $q = 1$	$O(n^3 \log n)$[72]
Voting Algorithm	$pq < 2/5$	$O(n \log n)$ (refer Chapter 7)

Table 2.2: Noisy OneMax with multiple bit-flipping prior noise.

This comparative study of the theoretical runtime bounds provide us an idea regarding the efficiency of the following algorithms. It is clear that the expected runtime of the crossover-based PCEA and our newly introduced voting algorithm (refer Chapter 7) is much lower than the other algorithms in presence of posterior noise. In case of multiple bit prior noise, the Voting algorithm has superior performance than the other algorithms for significant noise amount. These tables provide us a basis to choose a wide range of algorithms to study their performances in practice. We are interested to see if any of these algorithms with polynomial runtime bounds are actually helpful in practice for simple combinatorial problems like OneMax and Linear problems.

Algorithm	**Parameters**	**Runtime**	**n = 100** $\sigma = \mathbf{1}$	**n = 100** $\sigma = \mathbf{10}$
(1+1)–EA	ϱ =1/n,$\sigma^2 \leq (1/4\log(n))$	$O(n \log n)$ and super polynomial for larger σ^2 [51]	Super polynomial	Super polynomial
	ϱ =1/n,$\sigma^2 \geq 1$	$e^{\Omega(n)}$[96]	Exponential	Exponential
(1+1)–EA with Sampling	ϱ =1/n	$O(\sigma^2 n(\log n)^2)$[6]	$\sim$ 2121	$\sim$ 212076
Mutation-Population Algorithm		$O(\sigma^7 n \log n \log(\log n))$ [23]	$\sim$ 703	$\sim$ 7032922081
Paired Crossover EA (PCEA)	$\sigma^2 = n$	$O(n(\log n)^2)$ [91]		$\sim$ 2121
Compact GA (cGA)	$K = \omega(\sigma^2\sqrt{n}\log n)$	$O(K\sigma^2\sqrt{n}\log Kn)$ [47]	$\sim$ 3884	$\sim$ 60051903
Voting Algorithm	$\sigma^2 \leq 3n/8$	$O(n \log n)$	$\sim$ 461	
	$\sigma^2 > 3n/8$	$O(\sigma^2 \log n)$ [100] (refer Chapter 7)		$\sim$ 461

Table 2.3: Noisy OneMax with additive Gaussian noise. The calculations in the last two columns assume hidden constants = 1, for illustration purposes.

Most of the existing studies involve simple problems such as noisy OneMax or LeadingOnes problem. However, assessing the performance on algorithms on the simple problems might not yield a perfect guess for performance of EAs on other, more complex problems. We are motivated to look into more complex noisy combinatorial problems with constraints and multiple objectives as well.

The next chapters discuss more on the noise-handling capabilities of these most of these algorithms in practice, on solving un-constrained, constrained and multi-objective combinatorial problem when large noises are present.

Chapter 3

Simple Combinatorial Problems with Noisy Fitness Evaluations

The identification of the most efficient algorithm for noisy combinatorial optimisation would involve finding the algorithm that can solve a problem with a large level of noise in the least possible function evaluations. With this objective, firstly, we are interested to investigate whether any of the algorithms with the existing theoretical runtimes for noisy OneMax would be capable of solving combinatorial problems with added noise, in practice. A collection of algorithms on OneMax and Linear problems with posterior noise, are studied empirically, to see which algorithms can find optimal solutions within a reasonable number of function evaluations (to be defined below), bearing in mind that the asymptotic bounds for some of these algorithms, while polynomial, are actually very large (discussed in Chapter 2). We have considered posterior noise in this chapter to investigate the algorithm performances in large noise variances.

3.1 Algorithms Studied

Some algorithms in the chosen collection of EAs have been studied theoretically, while some are chosen here on the basis of their promise in solving noisy OneMax empirically. The following subsections illustrate the algorithm structures. The algorithms chosen are either mutation-based, crossover-based or EDAs, which would provide us an idea which evolutionary mechanism may be beneficial in handling large noises in practice.

3.1.1 The (1+1)–EA

The $(1 + 1)$–EA uses a mutation operator that produces an offspring by flipping each bit of the parent string independently with a probability of $1/n$. This can be considered as a randomised or stochastic hillclimber which considers only one point in the search space at a time and proceeds by trying to find a point which has a superior function value. In each iteration, only one function evaluation takes place.

Algorithm 1: The $(1 + 1)$–EA

```
Choose x ∈ {0,1}^n uniformly at random;
Set y = x;
while termination condition not reached do
    Flip each bit in y independently with probability 1/n;
    if f(y) ≥ f(x) then
        Set x = y;
    end
end
```

The expected runtime of the $(1 + 1)$–EA solving the non-noisy OneMax is $O(n \log n)$. The runtime remains polynomial in the posterior Gaussian noise case for $\sigma^2 < 1/(4 \log n)$, so we do not expect this algorithm to cope with anything but the smallest noise levels [51]. However, it would be an interesting experiment to identify at which level of noise the algorithm stops obtaining the optimum within a fixed budget of function evaluations. The structure of $(1 + 1)$–EA is presented in Algorithm 1.

3.1.2 Mutation-Population Algorithm

Since, it has long been recognised that populations can help an EA handle noise [55, 98], a mutation-based algorithm with a population is considered in this study. This non-elitist EA with tournament selection was analysed by Dang & Lehre to study how it optimises the noisy OneMax problem with posterior noise distributions [22, 23]. In case of additive Gaussian noise $N(0, \sigma^2)$ with mutation rate $\frac{\chi}{n} = \frac{a}{3\sigma n}$ and population size $\lambda = b\sigma^2 \ln n$ (where a and b are constants), the considered algorithm optimizes the OneMax problem in expected time $O(\sigma^7 n \ln(n) \ln(\ln(n)))$. It should be noted that this runtime bound is potentially very large, when the noise is large — in excess of $n^{4.5}$ when $\sigma = \sqrt{n}$, although of course this is an upper bound, and we do not know the constants.

Algorithm 2: The Mutation-Population Algorithm

Initialise population $\mathcal{P}$ uniformly at random, where $|\mathcal{P}| = \lambda$;
while *termination condition not reached* **do**
    Set population of next generation, $\mathcal{P}' = \Phi$;
    repeat λ **times**
        Sample two parents $x, y \sim \text{Uniform}(\mathcal{P})$;
        if $f(x) > f(y)$ **then**
            Set $z = x$;
        else if $f(x) < f(y)$ **then**
            Set $z = y$;
        else
            Sample $z \sim \text{Uniform}(\{x, y\})$;
        end
        Flip each bit in z independently with probability $1/n$;
        Set $\mathcal{P}' = \mathcal{P}' \cup z$;
    end
    Set $\mathcal{P} = \mathcal{P}'$;
end

It should also be observed that the noise variance needs to be known in advance to run this algorithm, since the algorithm parameters like mutation rate and population size are dependent on the noise variance. In realistic scenarios, the exact estimation of noise variance is difficult.

3.1.3 Compact Genetic Algorithm (cGA)

The compact GA (cGA) is an EDA, introduced by Harik et al. [58]. The upper runtime bound of cGA for noisy OneMax problem with Gaussian noise of variance σ^2 is given as $O(K\sigma^2\sqrt{n}\log Kn)$ when $K = \omega(\sigma^2\sqrt{n}\log n)$, with probability $1 - o(1)$. It is to be noted that this upper bound is in excess of n^3 when $\sigma = \sqrt{n}$. According to the runtime results, the value of σ needs to be known in advance to set the algorithm parameters.

Algorithm 3: The Compact Genetic Algorithm (cGA)

```
Initialise probability vector p = (0.5, 0.5, ..., 0.5);
while termination condition not reached do
    Generate search strings x and y from vector p;
    if f(x) < f(y) then
        Swap x and y;
    end
    for i ∈ {1, 2, ..., n} do
        if x_i > y_i then
            Set p_i ← p_i + 1/K;
        else if x_i < y_i then
            Set p_i ← p_i − 1/K;
        else
            p_i = p_i;
        end
    end
end
```

3.1.4 Population Based Incremental Learning (PBIL)

The algorithm PBIL, proposed by Baluja [9] in 1994, combines genetic algorithms and competitive learning for optimising a function. We have included this algorithm as it is in some ways similar to the cGA, so we might expect it to have similar performance. We are not aware of any theoretical analysis of this algorithm on noisy problems. The runtime of PBIL on OneMax (with no noise) is known to be $O(n^{3/2}\log n)$, for suitable choice of λ [120].

Algorithm 4: Population-Based Incremental Learning (PBIL)

Initialise probability vector $p = (0.5, 0.5, \ldots, 0.5)$;
while *termination condition not reached* **do**
  for $j \in \{1, 2, \ldots, \lambda\}$ **do**
    Sample search string x^j from p and evaluate $f(x^j)$;
  end
  Sort the population $\mathcal{P} \leftarrow \{x^1, x^2, \ldots, x^\lambda\}$ according to fitness;
  Update probability vector according to μ fittest individuals, where η is smoothing parameter

$$p_i \leftarrow (1-\eta)p_i + (\eta/\mu)\sum_{j=1}^{\mu} y_i^j \quad \forall i \in \{1, 2, \ldots, n\}, y^j \in \mathcal{P}_{sorted};$$

end

3.1.5 Univariate Marginal Distribution Algorithm (UMDA)

The Univariate Marginal Distribution Algorithm (UMDA) proposed by Mühlenbein [81] belongs to the EDA schema. In some ways, it is therefore similar to cGA and PBIL. However, it can also be viewed as generalising the *genepool* crossover scheme, in which the bits are shuffled across the whole population (within their respective string positions). We have included UMDA therefore, to see if its behaviour is more like cGA and PBIL on the one hand (which emphasise an evolving distribution over bit values), or like PCEA on the other (which emphasises crossover). We have also observed that UMDA has certain resilience to noise in OneMax problem from previous literature [87].

The UMDA algorithm initialises a population of λ uniformly random solutions, and sorts the population according to the fitness evaluation of each candidate solution. The best μ members of the population are selected. These are used to calculate the sample distribution of bit values in each position. The next population is generated from this distribution in a similar manner as PBIL (with $\eta = 1$).

There are two variants of UMDA, depending on whether the probabilities are constrained to stay away from the extreme values of 0 and 1, or not. It is known

Algorithm 5: Univariate Marginal Distribution Algorithm (UMDA)

Initialise probability vector $p = (0.5, 0.5, \ldots, 0.5)$;
while *termination condition not reached* **do**
  for $j \in \{1, 2, \ldots, \lambda\}$ **do**
    Sample search string x^j and evaluate $f(x^j)$;
  end
  Sort the population $\mathcal{P} \leftarrow \{x^1, x^2, \ldots, x^\lambda\}$ according to fitness;
  Update probability vector according to μ fittest individuals,

$$p_i \leftarrow \frac{1}{\mu} \sum_{j=1}^{\mu} y_i^j \qquad \forall i \in \{1, 2, \ldots, n\}, y_i \in \mathcal{P}_{sorted};$$

end

that if the population size is large enough (that is, $\Omega(\sqrt{n} \log n)$) then this handling of the probabilities at the margins is not required [119]. Since we will work with a large population (to match the PCEA algorithm described below), we will not employ margin handling. In our experiments we will take $\mu = \lambda/2$. We are not aware of any theoretical results concerning UMDA on problems with posterior noise, but the runtime on OneMax (with no noise) is known to be $O(n \log n)$ for $\mu = \Theta(\sqrt{n} \log n)$ [119].

3.1.6 Paired-Crossover EA (PCEA)

Recently, the recombination operator has been suggested to be considerably beneficial in noisy evolutionary search. Prügel-Bennett et al. [91] considered the problem of solving OneMax with noise of order $\sigma = \sqrt{n}$ and analysed the runtime of an evolutionary algorithm consisting only of selection and uniform crossover, the Paired-Crossover EA (PCEA). They show that if the population size is $c\sqrt{n} \log n$ then the required number of generations is $O\left(\sqrt{n} \log n\right)$, giving a runtime of $O(cn\left(\log n\right)^2)$, with the probability of failure is $O(1/n^c)$. It is not known what happens for lower levels of noise, though it is shown that in the absence of noise, PCEA solves OneMax in $O(n(\log n)^2)$, which matches the runtime bound (up to constant factors)

Algorithm 6: Paired-Crossover EA (PCEA)

Initialise population $\mathcal{P}$ uniformly at random, where $|\mathcal{P}| = \lambda$;
Set next generation of population, $\mathcal{P}' = \Phi$;
while *termination condition not reached* **do**
    repeat λ **times**
        Choose two parents α and β at random from $\mathcal{P}$;
        Apply uniform crossover to create children x and y;
        if $f(x) > f(y)$ **then**
            Add x to $\mathcal{P}'$, $\mathcal{P}' = \mathcal{P}' \cup x\}$ **else**
                Add y to $\mathcal{P}'$, $\mathcal{P}' = \mathcal{P}' \cup y$;
            end
    end
    Set $\mathcal{P} = \mathcal{P}'$;
end

for the case $\sigma = \sqrt{n}$.

3.2 Parameter Choices for Experiments

We investigate the performance of the algorithms described above, in solving the noisy OneMax problem. When the fitness evaluation is tampered with additive Gaussian noise, the fitness function of OneMax becomes as follows,

$$\text{Noisy}\textsc{OneMax}(x) = \sum_{i=1}^{n} x_i + N(0, \sigma)$$

Since fitness evaluation is the most relevant step in selecting a better solution, we expect the performance of all the algorithms to be affected by noise. We are interested to know if the performances of the algorithms are helpful in practice, in correspondence to the existing theoretical proofs for the expected polynomial runtime of specific algorithms on solving the noisy OneMax problem with additive posterior Gaussian noise [91, 22, 6, 47, 94, 24, 32, 28].

We investigate the algorithms' performances given a reasonable but fixed run-time budget across a wide range of noise levels, from $\sigma = 0$ up to $\sigma = \sqrt{n}$ in the first instance. We also investigate the performance of the better performing algorithms for much larger noise variances extending up to $\sigma = n$ and observe the best solution (though not optimal) found and the number of function evaluations needed to find it, by these algorithms.

To address the question of what constitutes a *reasonable* budget, we compared the known theoretical results of our algorithms on noisy OneMax. PCEA has the lowest proven upper bound on its runtime, compared to the other algorithms for which results exist. We therefore allowed each algorithm to have twice the number of fitness evaluations that PCEA requires (on average) to find the optimum, as a reasonable budget. The function evaluation budgets calculated in this way are given in Table 3.3.

The population size for the PCEA is taken to be $10\sqrt{n}\log n$ according to the theoretical proofs and empirical study by [91]. According to the proofs by [22], the population size $\lambda = \sigma^2 \log n$ is chosen for the Mutation-Population algorithm. According to the paper by [47], the parameter $K = 7\sigma^2\sqrt{n}(\log n)^2$ is considered for cGA. In presence of additive posterior noise, PBIL and UMDA have not yet been studied much. For PBIL, the population size is taken as $\lambda = 10n$ (following the theoretical requirement of [120]). From these, we select the best $\mu = \lambda/2$ individuals. In case of UMDA, the total number of generated candidates in a particular generation is chosen as $20\sqrt{n}\log n$, so that the effective population size is the same as for PCEA. All these parameter settings are retained for all of our experiments in simple and constrained noisy combinatorial optimisation problems (in next chapter).

3.3 Experiments on Noisy OneMax Problem

The Figure 3.1 illustrates a comparison of all of the considered algorithms while solving the noisy OneMax problem for problem size $n = 100$. Here, the aver-

Algorithm	**Noise Levels**	**Parameters**
PCEA	$\sigma = 1$–10	$\lambda = 461$
(1+1)–EA	$\sigma = 1$–10	$\sim$
Mutation-Population	$\sigma = 1$, $\sigma = 2$, $\sigma = 3$, $\sigma = 4$, $\sigma = 5$, $\sigma = 6$. $\sigma = 7$, $\sigma = 8$, $\sigma = 9$, $\sigma = 10$	$\lambda = 5$, $\lambda = 18$, $\lambda = 41$, $\lambda = 74$, $\lambda = 115$, $\lambda = 166$, $\lambda = 226$, $\lambda = 295$, $\lambda = 373$, $\lambda = 461$
PBIL	$\sigma = 1$–10	$\lambda = 1000$
UMDA	$\sigma = 1$–10	$\lambda = 921$
cGA	$\sigma = 1$, $\sigma = 2$, $\sigma = 3$, $\sigma = 4$, $\sigma = 5$, $\sigma = 6$. $\sigma = 7$, $\sigma = 8$, $\sigma = 9$, $\sigma = 10$	$K = 1485$, $K = 5938$, $K = 13361$, $K = 23753$, $K = 37113$, $K = 53443$, $K = 72742$, $K = 95010$, $K = 120247$, $K = 148453$

Table 3.1: Parameter choices for the experiments with $n = 100$.

age performance of 100 runs are illustrated. Different levels of Gaussian additive noise with mean 0 and standard deviation $\sigma = 1$ to 10 are considered in the first experiment.

It can be seen that PCEA and UMDA are resistant to these noise levels as they are capable of finding the global optimum within the given budget. The runtimes for these two algorithms are shown in Figure 3.2. It can be seen from the figure that the runtimes of these algorithms scale with the noise variance, although they

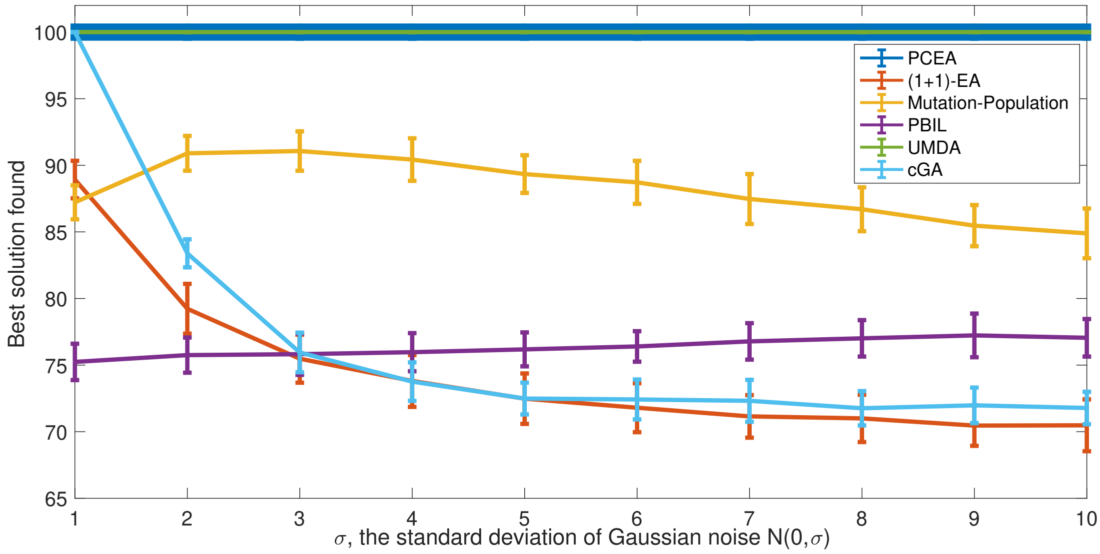

Figure 3.1: Comparison of the algorithms while solving the noisy OneMax for different noise levels

manage to find the optimum.

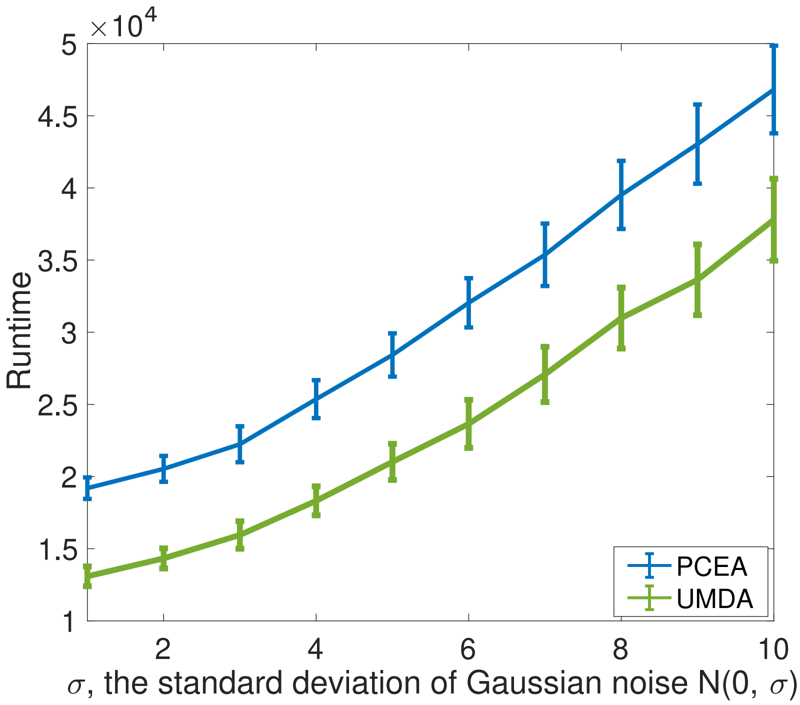

Figure 3.2: Runtime comparison of UMDA and PCEA for noisy OneMax

However, the mutation-based algorithms $(1+1)$–EA and Mutation-Population algorithm, cannot cope with these small levels of noise. Interestingly, the performance of Mutation-Population slightly improves initially with noise variance (which may be likely due to the increase of diversity within the population arising from noisy fitness evaluations, where worse solutions are retained), however consistently becomes worse as the noise variance increases further.

σ	1	2	3	4	5
budget	38392	41066	44477	50728	56851
σ	6	7	8	9	10
budget	64079	70736	79034	86078	93638

Table 3.2: Function evaluation budgets allowed for noisy OneMax experiments with different noise levels.

It can be observed that PBIL and cGA are not able to cope with even these small levels of noise within the given fixed budget of function evaluations. Although, UMDA, PBIL and cGA are all EDAs, it is interesting to observe that only UMDA is resilient to noise upto the variance of $\sigma = \sqrt{n}$. It seems likely that the improved performance of UMDA is due to the stronger selection mechanism (truncation selection) and its close resemblance to genepool crossover.

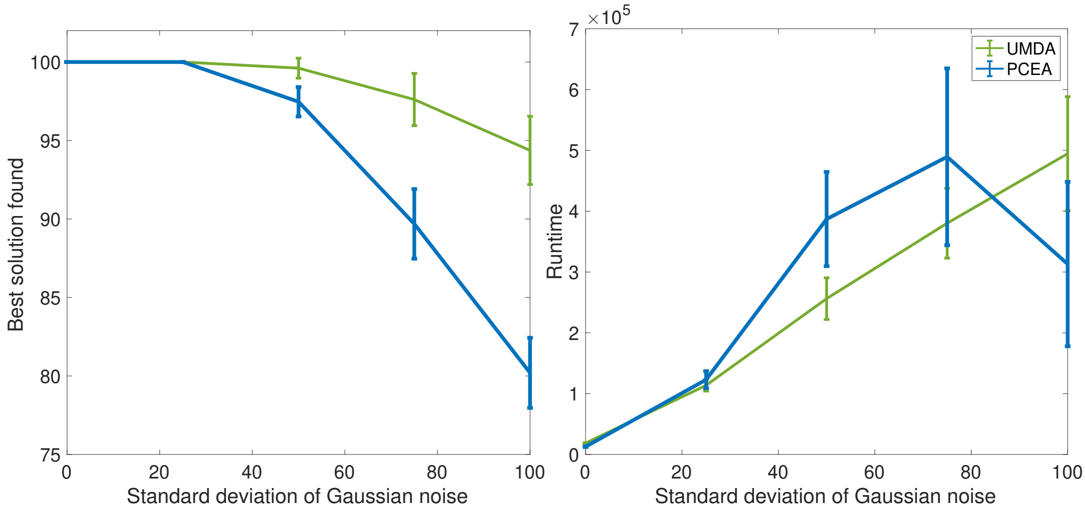

Figure 3.3: Performance comparison of UMDA and PCEA for noisy OneMax with very large levels of noise

For larger noise variances, we investigate the best solution found and the function evaluations required to do so, for the better performing algorithms, UMDA and PCEA. Figure 3.3 compares the performance of these two algorithms. For this experiment as well, PCEA is run until the population converges and UMDA is run for twice the number of function evaluations needed by PCEA. It is interesting to note that for problem size $n = 100$, both PCEA and UMDA can cope with noise variances upto almost one-fourth of the problem size. Beyond that, for PCEA, the best solution found until the population converges, is not the optimum. For UMDA, the best solution found is slightly better than PCEA, within the fixed function evaluation budget.

The Mann-Whitney U-test is performed on the samples of best results achieved and the runtime of the algorithms, with the null hypothesis that they are from distributions with equal medians. For each data point, the null hypothesis is rejected at 5% significance level. The results of the Mann Whitney U-test confirm that the performance of UMDA is slightly better than PCEA, however both of these algorithms can tolerate noise well.

3.4 Experiments on Noisy Linear Problem

The LINEAR problem with n positive weights $w_1, \dots, w_n$ and the corresponding noisy variant are defined as,

$$\textsc{Linear}(x) = \sum_{i=1}^{n} x_i w_i$$
$$\textsc{NoisyLinear}(x) = \sum_{i=1}^{n} x_i w_i + N(0, \sigma)$$

In generating random problem instances, we draw the weights uniformly at random from the range $1, \dots, 100$. Thus we avoid more extreme instances such as BINVAL (in which $w_i = 2^{i-1}$ for each $i = 1, \dots, n$). The reason for this is that when the distribution of weights is highly skewed, the addition of noise is irrelevant for those bits with very high weights, yet completely overwhelms bits with weights lower than the typical noise level. Thus most algorithms will find the more significant bits, and fail on the remaining bits.

σ	1	2	3	4	5
budget	47096	46801	47704	48350	48682
σ	6	7	8	9	10
budget	49954	50876	51429	52794	53310

Table 3.3: Function evaluation budgets allowed for noisy LINEAR experiments

The maximisation of the LINEAR problem as defined above has only one global optimum, which is the sum of all the weights. The ONEMAX problem is a special case of the LINEAR problem when all of the weights are units. However, optimising the LINEAR problem is more difficult than ONEMAX, as the bits with heavier weights get optimised with a higher preference than the bits with lower weights.

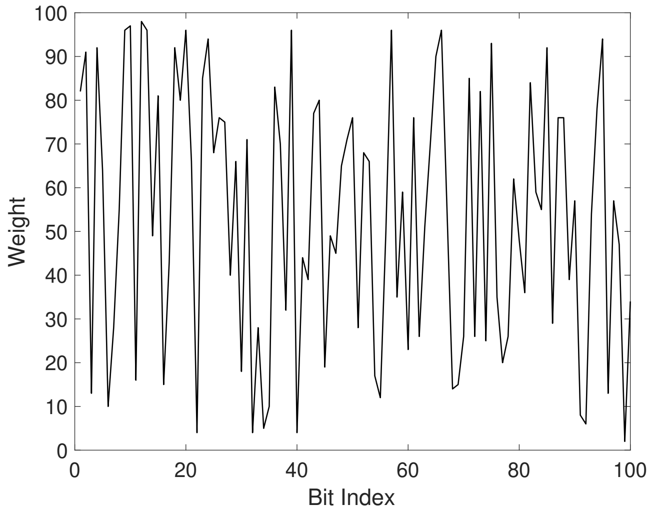

Figure 3.4: A typical problem instance for the LINEAR problem considered in the experiments

Random problem instances were studied with 100 randomly chosen weights between 1 and 100. A typical randomly chosen instance for the LINEAR problem is illustrated in Figure 3.4. The variance of the randomly chosen weights with respect to the indices of the bits is illustrated. This provides an idea regarding the problem difficulty.

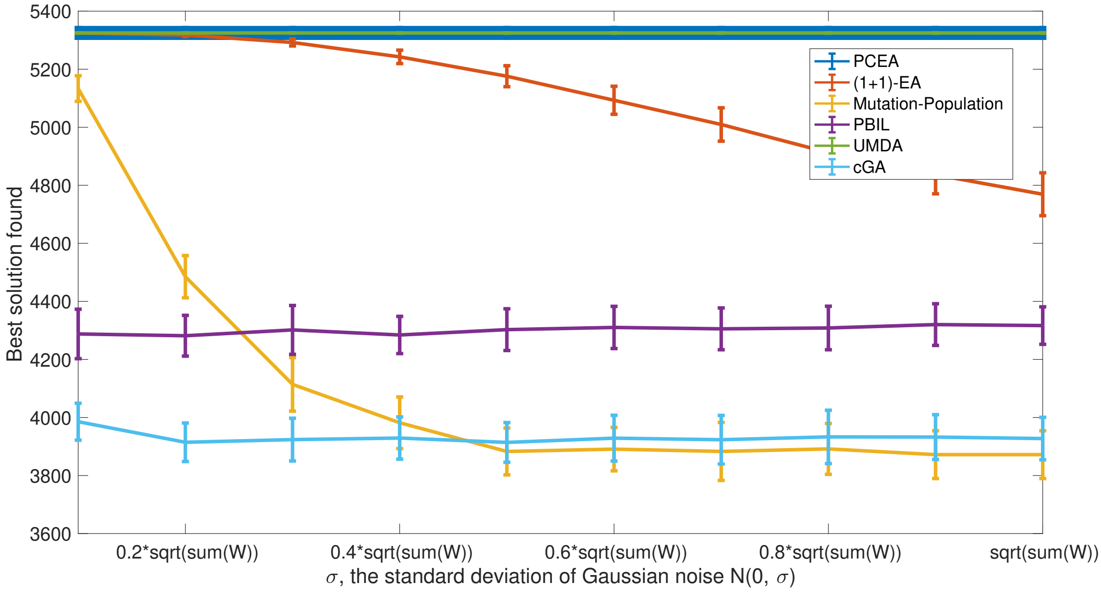

Figure 3.5: Comparison of algorithms while solving noisy WeightedLinear

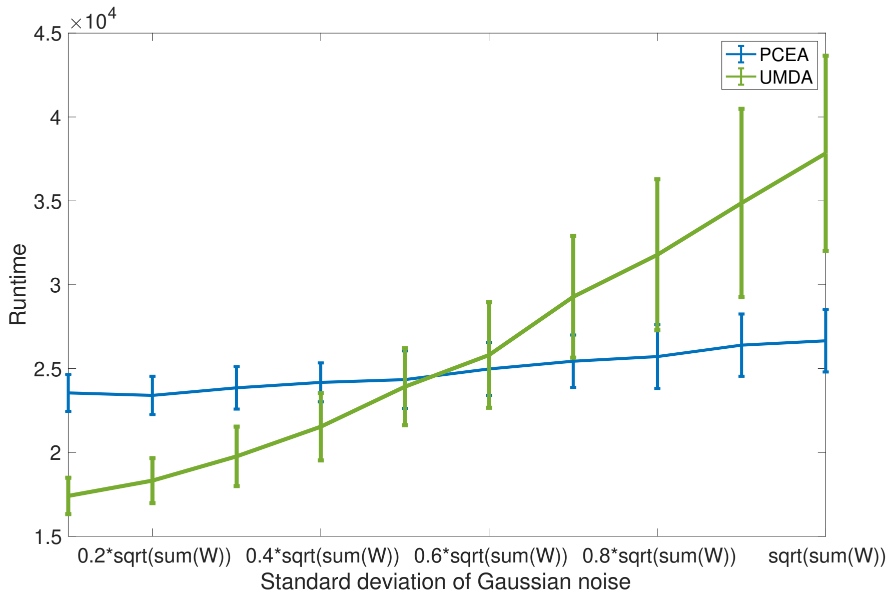

Figure 3.6: Runtime comparison of UMDA and PCEA for noisy WeightedLinear

The results for this instance of Linear problem are shown in Figure 3.5 with averages over 100 runs. Since the optimum of the problem would be sum of all the weights, the noise variances are chosen in terms of the optimum. The standard deviation of the Gaussian noise is shown as multiples of the square root of the sum of the weights. The function evaluation budget allowed to each of the algorithms are fixed at twice the average runtime of PCEA at each noise level (see Table 3.3).

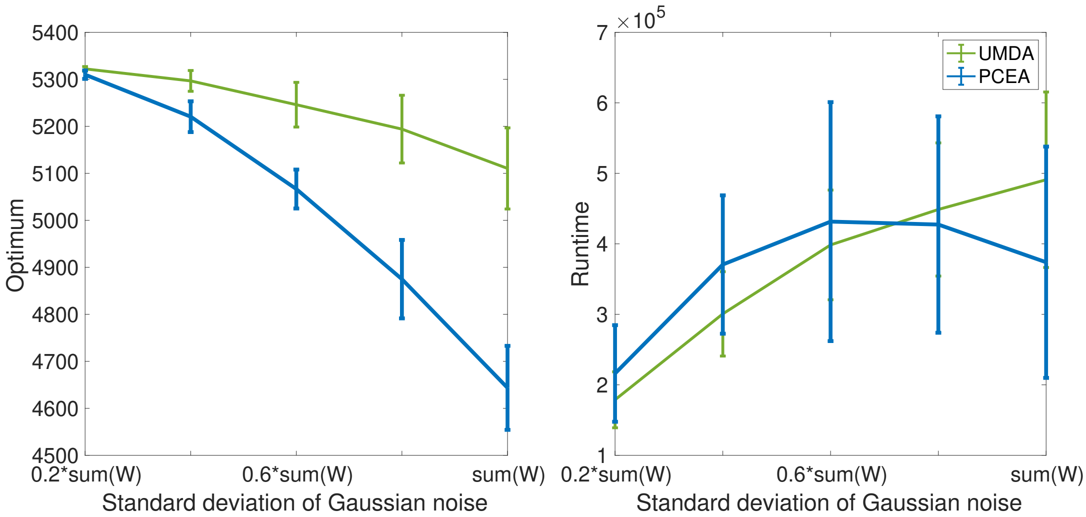

Figure 3.7: Performance comparison of UMDA and PCEA for noisy WEIGHTED-LINEAR with very large levels of noise

The plot in Figure 3.5 illustrates the performance comparison of all of the considered algorithms while solving the noisy LINEAR problem for the problem size $n = 100$ with noise variances upto square root of the sum of all the weights. As evident from Figure 3.5, the curves of PCEA and UMDA are coincident, showing that they can cope with the noise well and are resistant up to these levels of noise. The runtime of UMDA and PCEA are plotted in Figure 3.6. However, the performance of the $(1+1)$–EA and Mutation-Population algorithm worsen with increasing noise. The small step sizes of $(1+1)$–EA help slightly in optimising the LINEAR problem which is clearly not present in the effect of population in Mutation-Population algorithm. Even with relatively small noise levels, the cGA and PBIL are not able to solve the problem within twice the runtime of PCEA.

The better performing algorithms, PCEA and UMDA are studied for even larger levels of noise. The Figure 3.7 illustrates the performance of these two algorithms for noise variances as large as the sum of all the weights.

The Mann Whitney U-test is performed for all the experiments with the noisy LINEAR problem. The slight better performance of UMDA in this problem is statistically significant, however, both UMDA and PCEA can tolerate high levels of noise in the LINEAR PROBLEM.

3.5 Summary

In this chapter, we experimentally compared a collection of algorithms on noisy ONEMAX and noisy LINEAR problems, to see which can find solutions within a reasonable function evaluations budget. The asymptotic bounds of some of these algorithms are polynomial in nature, however are very large. We investigated the empirical performance and identified that, in practice most of these algorithms are not capable to handle noise.

However, it is evident from the empirical results of these simple noisy problems that uniform crossover-based PCEA and UMDA can cope with noise significantly better than the other algorithms. At this point, it is interesting to note that, UMDA employs a mechanism similar to *genepool crossover*, where at each bit position, the offspring bit is obtained by recombination of that bit across the whole parent population. It is hypothesised that these two algorithms are therefore highly similar in operation. The next chapter investigates these better performing algorithms further for harder combinatorial problems (without and with constraints) in presence of noise.

Chapter 4

Harder Noisy Combinatorial Problems

The implication of the success of UMDA and PCEA on the noisy toy problems, and the failure of the other algorithms to cope with even modest levels of noise, provides the basis to move to the second stage of the study considering only UMDA and PCEA. In this chapter, these two better performing algorithms are chosen and investigated how well they can handle noise in three combinatorial problems: SubsetSum, Knapsack and SetCover. We choose these, as they have a 'packing' structure which might make them amenable to algorithms which can solve noisy OneMax efficiently. We generate random problem instances within the 'easy' regime (so that the algorithms can be expected to solve them when there is no noise) and then empirically study how they degrade with added Gaussian noise.

4.1 Noisy SubsetSum Problem

The SubsetSum problem is defined with reference to n positive weights $w_1, \ldots, w_n$ and a target θ as follows,

$$\textsc{SubsetSum}(x) = \left| \theta - \sum_{i=1}^{n} x_i w_i \right|$$

This combinatorial problem tries to find the best combination of weights as close as possible to the target. The search string is defined such that, it has a one in the bit position i if the ith weight is chosen. The SubsetSum has several real world applications such as packing problems.

In presence of noisy fitness evaluations where the noise is additive Gaussian in nature, the fitness function can be written as follows,

$$\text{NoisySubsetSum}(x) = \left|\theta - \sum_{i=1}^{n} x_i w_i\right| + N(0, \sigma)$$

Here, the mean of the noise is assumed to be zero.

The SubsetSum problem can be seen as a generalisation of the Linear problem (in which the target is $\theta = 0$). In our experiments, we generate instances by choosing positive weights uniformly at random from $1, \ldots, 100$. We consider the target to be two-thirds of the sum of the weights, given as,

$$\theta = \frac{2}{3} \sum_{i=1}^{n} w_i$$

We have run experiments for other choices of θ and found that they do not significantly affect the empirical observations.

For the experiments, a range of four problem sizes is considered with 50, 100, 150 and 200 positive weights, each lying between 1 and 100, and chosen uniformly at random. Corresponding to each of the four problem sizes, 10 different problem instances are considered, i.e., a total of forty problem instances are considered. The target θ is considered to be two-third of the sum of all the weights for each respective problem instance. The additive Gaussian noise considered in the SubsetSum problem is centered at zero. The standard deviation of the noise is considered as the integral multiples of the mean of the weights, viz., $5 \times mean(W)$, $10 \times mean(W)$, $15 \times mean(W)$ and $20 \times mean(W)$.

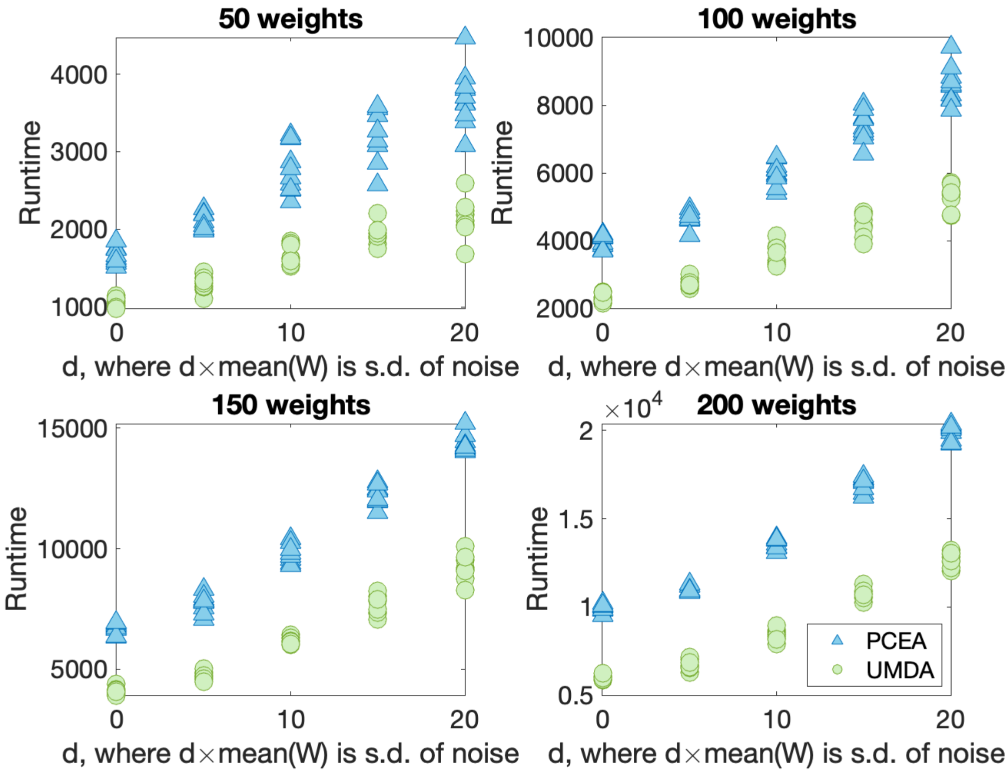

Figure 4.1: Runtime comparison of UMDA (blue circles) and PCEA (green triangles) while solving forty instances of the noisy SubsetSum problem.

The noisy SubsetSum problem being a minimisation problem, if we obtain the (non-noisy) fitness value of zero, we obtain the global optimum. Both of the algorithms are able to find the global optimum for all of these problems and their corresponding noise levels. Each algorithm is run until the global optimum is reached. We therefore plot the runtime (the number of function evaluations required to reach the global optimum) against the standard deviation of the noise — please refer Figure 4.1. Each data point in the scatter plot represents the number of function evaluations (averaged over 100 runs) required to reach the global optimum for each problem instance. It is evident that the runtime of the algorithm scales with the problem size and noise variance. In order to obtain the statistical significance of results, the Mann-Whitney U-test is used. It is observed that UMDA has a slightly better runtime than PCEA.

4.2 Noisy Knapsack Problem (Version 1)

The KNAPSACK problem is defined with respect to a set of weights $w_1, \ldots, w_n$, and their respective profits $p_1, \ldots, p_n$ and the capacity of the Knapsack, C, as follows,

$$\text{KNAPSACK}(x) = \begin{cases} \sum_{i=1}^{n} x_i p_i & \text{if } \sum_{i=1}^{n} x_i w_i \leq C \\ C - \sum_{i=1}^{n} x_i w_i & \text{otherwise} \end{cases}$$

Here, the weights and their profits are positive integers. We aim to maximise the sum of the profits of the weights added to the Knapsack by keeping in mind that the capacity of the Knapsack is not exceeded.

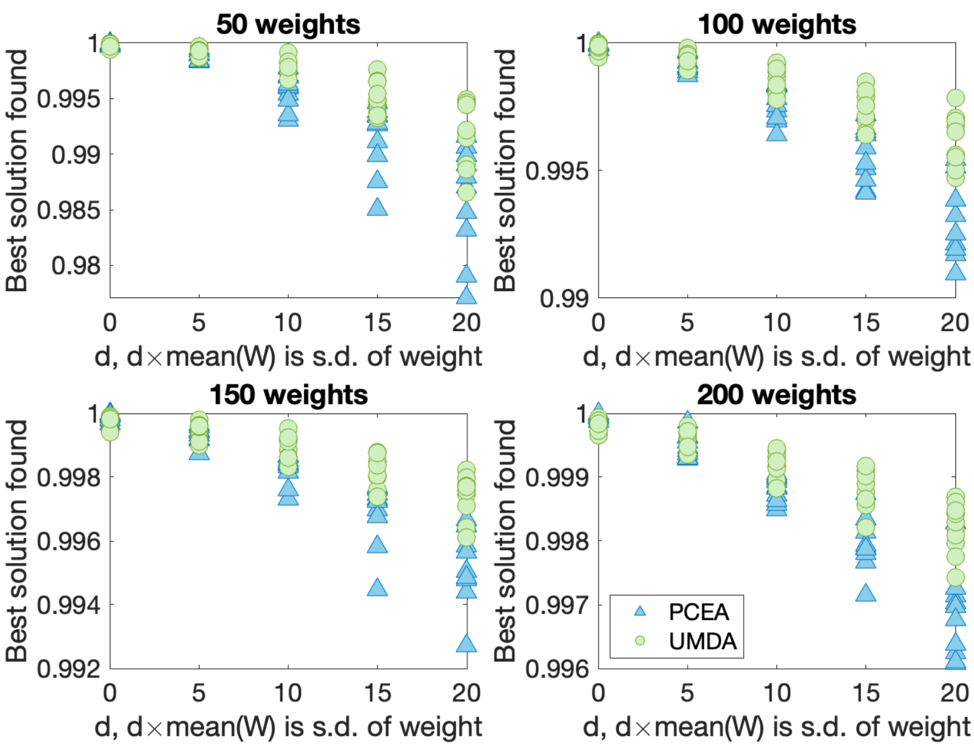

Figure 4.2: Solution quality of UMDA (circles) and PCEA (triangles) while solving instances of NOISYKNAPSACKV1

Random problem instances are chosen with weights and profits belonging to the interval $[1, 100]$. The capacity of the Knapsack is set to two-thirds of the sum of the weights. We consider two noisy variants of the Knapsack problem. The first version simply considers posterior additive noise as before:

$$\text{noisy}\textsc{Knapsack}\text{V1}(x) = \textsc{Knapsack}(x) + N(0, \sigma)$$

For the first version of the noisy Knapsack problem, instances with 50, 100, 150 and 200 weights are considered. The weights are randomly chosen between 1 and 100, with associated profits chosen in the same range. The maximum capacity of the knapsack C is considered to be two-thirds of the sum of all the weights considered.

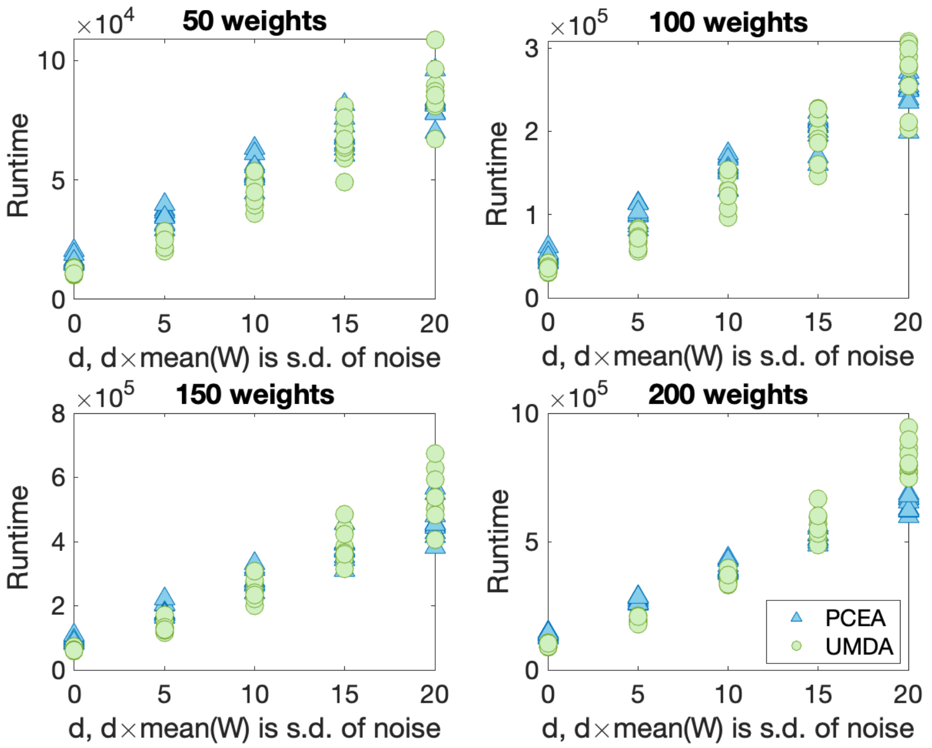

Figure 4.3: Comparison of runtime of UMDA (circles) and PCEA (triangles) while solving the NoisyKnapsackV1

When noise is added, neither algorithm finds the optimal solution, so we record the best solution found (as assessed by non-noisy fitness function). PCEA is run until the population converges whereas, UMDA is run for twice that time, and we report the time taken to find the best solution encountered.

For each problem instance, we plot (in Figure 4.2) the best solution found (averaged over 100 runs) as a fraction of the best solution ever encountered for that problem instance. This enables us to make meaningful comparisons between problem instances. The best known solution for each problem instance has a scaled fitness value of 1. For each of the four problem sizes, 10 different problems instances are considered.

Figure 4.3 shows the time taken (on average) to locate the best found solution in each case, illustrated as runtime in this case. We can observe in Figures 4.2 and 4.3, that both the algorithms can find good, though not optimal solutions, for NoisyKnapsackV1 with significant levels of noise. The observations from Mann-Whitney U-test show that UMDA is slightly better than PCEA with these parameter settings.

4.3 Noisy Knapsack Problem (Version 2)

In the second version, the presence of noise in the judgement with respect to the weights is considered,

$$W_\sigma(x) = \sum_{i=1}^{n} x_i w_i + N(0,\sigma)$$

If this (noisy) weight does not exceed the capacity, we then evaluate (noisily), the profit. Otherwise we return the excess weight:

$$\textsc{NoisyKnapsackV2}(x) =$$

$$\begin{cases} \sum_{i=1}^{n} x_i p_i + N(0,\sigma) & \text{if } W_\sigma(x) \leq C \\ C - W_\sigma(x) & \text{otherwise} \end{cases}$$

Note that noise is added to the weight just once, when the constraint is checked, and the same value used to report the fitness value, in the case the constraint is violated.

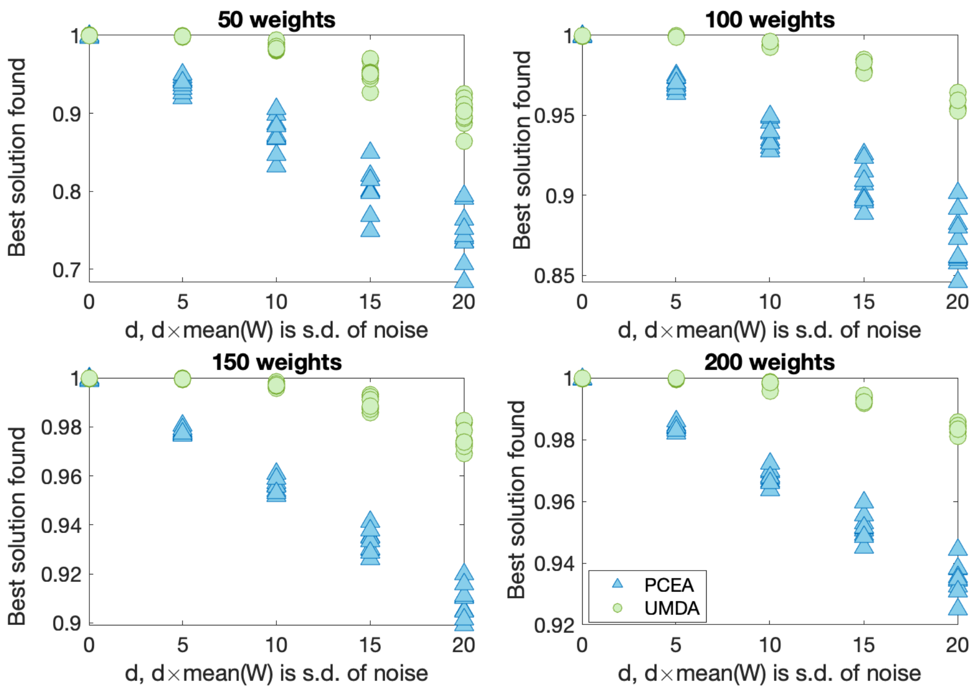

Figure 4.4: Solution quality of UMDA (circles) and PCEA (triangles) while solving the NoisyKnapsackV2

When the measurements of the weights is uncertain, as well as the profits, this creates a more complex noise model for the Knapsack problem. In the first stage, the total weight of the proposed solution is compared against the capacity, and this is done with added noise. Hence it may be thought that the proposed solution is feasible when in fact it is not. If it is considered feasible, then the benefit (total profit) is calculated, again with added noise.The parameters are considered as in the previous version of the Knapsack problem. 10 problems each of 50, 100, 150, and 200 weights (lying between 1 and 100) with associated profits (also lying in the same range), i.e., forty problems in total are considered.

Figure 4.4 depicts how the best (non-noisy) solution varies for different problem

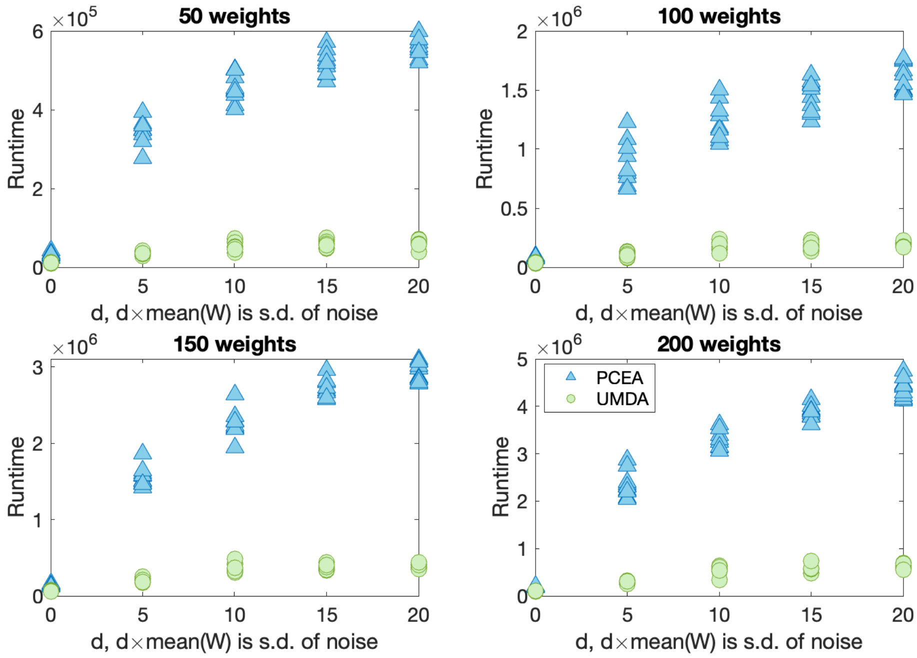

Figure 4.5: Comparison of runtime of UMDA (circles) and PCEA (triangles) while solving the NoisyKnapsackV2

sizes. This value is scaled with respect to the best value found when there is no noise. PCEA is run until the population converges while UMDA is run for twice that time, and we report the time taken to find the best solution encountered in Figure 4.5. The Mann-Whitney U-test shows that the best solution achieved and corresponding runtime of UMDA is better than PCEA in these particular parameter settings. The runtime required to find these values is shown in Figure 4.5, and we see that UMDA finds its best solution considerably faster than PCEA.

4.4 Noisy ConstrainedSetCover and PenaltySetCover Problems

The SetCover problem finds a minimal covering of m elements with a collection of sets from n pre-defined subsets. A Boolean matrix a_{ij} with n-rows and m-columns is used to define the n subsets $c_1, \ldots, c_n$:

$$a_{i,j} = [i \in c_j]$$

Here, we use the convention that $[expr]$ equals 1 if $expr$ is true, and 0 otherwise. The optimal collection of the sets would have the least number of the sets needed to cover all the m elements. The SetCover problem has several real-world applications such as the airline crew scheduling problem. The problem can be defined as a constrained single-objective one, as well as, a single-objective problem with a penalty term. The problem can also be defined as a multi-objective problem (discussed later).

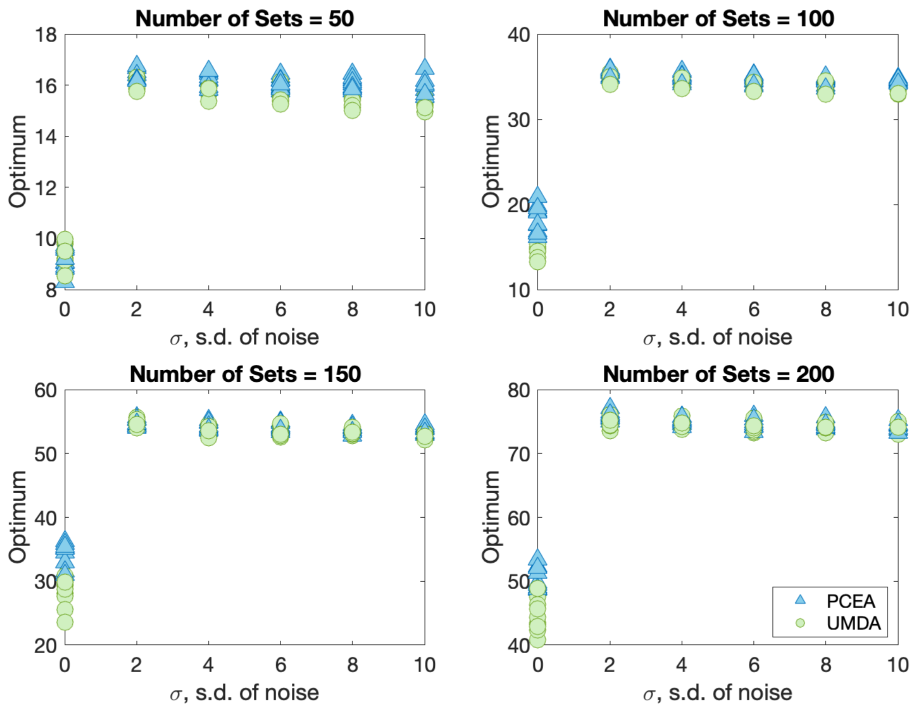

Figure 4.6: Solution quality of UMDA (circles) and PCEA (triangles) while solving the ConstrainedSetCover

The ConstrainedSetCover problem has a constraint that checks if the solution covers each of the m elements. The optimal solution would have the least number of sets needed to cover all the m elements. It is defined as follows,

$$\textsc{ConstrainedSetCover}(x) = \sum_{j=1}^{n} x_j$$

$$\text{subject to} \sum_{j=1}^{n} x_j a_{ij} \geq 1,\ \ i \in 1, \ldots, m$$

For comparison-based algorithms, we always prefer feasible solutions instead of infeasible solutions. Two feasible solutions are compared by their fitness values, whereas two infeasible solutions by their constraint violations. The noisy version of the problem arises if the judgements regarding the number of elements uncovered and the number of the subsets required is noisy. The NoisyConstrainedSetCover problem with additive posterior noise can be defined as follows.

$$\textsc{NoisyConstrainedSetCover}(x) = \sum_{j=1}^{n} x_j + N(0, \sigma)$$

$$\text{subject to} \sum_{j=1}^{n} x_j a_{ij} + N(0, \sigma) \geq 1,\ \ i \in 1, \ldots, m$$

The ConstrainedSetCover problem is solved by initially finding the feasible solutions and then minimising the number of the selected sets. This lexicographic ordering is achieved in the selection mechanism of the considered algorithms.

The algorithm structure of PCEA and UMDA is considered in a similar manner as the previous experiments. In PCEA, the child with least uncovered elements is selected. When both of the children have the same number of uncovered elements, the child with the minimum number of sets goes to the next population. In UMDA, the sorting of the population is based on the above mentioned lexicographic ordering. We consider margin handling in UMDA for all the following experiments in the SetCover problem due to increased problem difficulty.

The fitness function of SetCover problem can also be defined by including a penalty term such that, if elements are under-covered by the considered collection of sets, a huge penalty μ is incurred.

$$\textsc{PenaltySetCover}(x) =$$

$$\sum_{j=1}^{n} x_j + \mu \sum_{i} \max \left\{ 0, \left(1 - \sum_{j=1}^{n} a_{ij} x_j \right) \right\}$$

This gives rise to a corresponding noisy variant:

$$\textsc{NoisyPenaltySetCover}(x) =$$

$$\sum_{j=1}^{n} x_j + \mu \sum_{i} \max \left\{ 0, \left(1 - \sum_{j=1}^{n} a_{ij} x_j \right) \right\} + N(0, \sigma)$$

The alternative PenaltySetCover problem handles the constraint within the penalty function, hence creating a single objective.

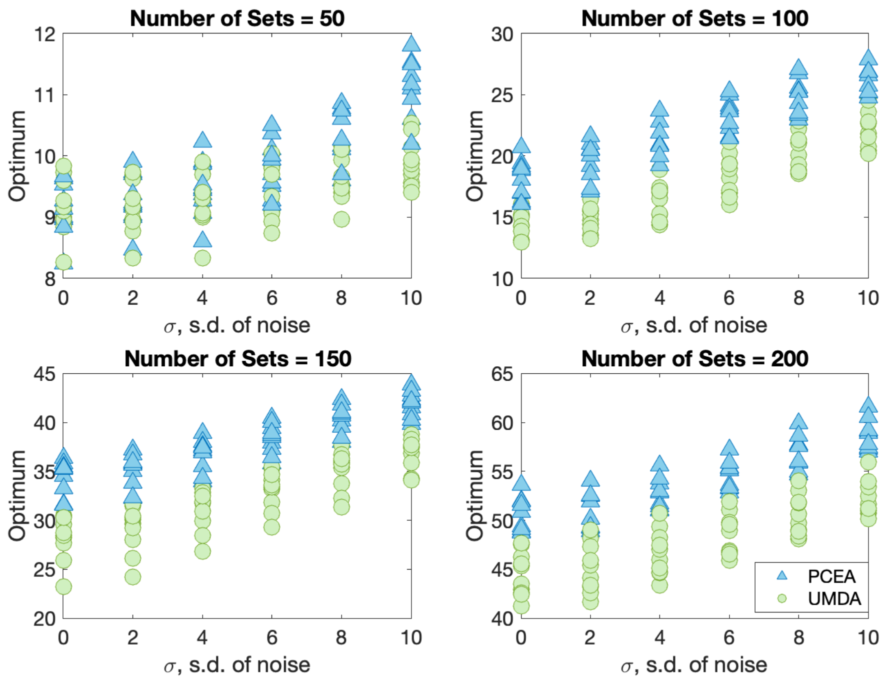

Figure 4.7: Best solution found in stipulated budget of function evaluations by UMDA (circles) and PCEA (triangles) for NoisyPenaltySetCover

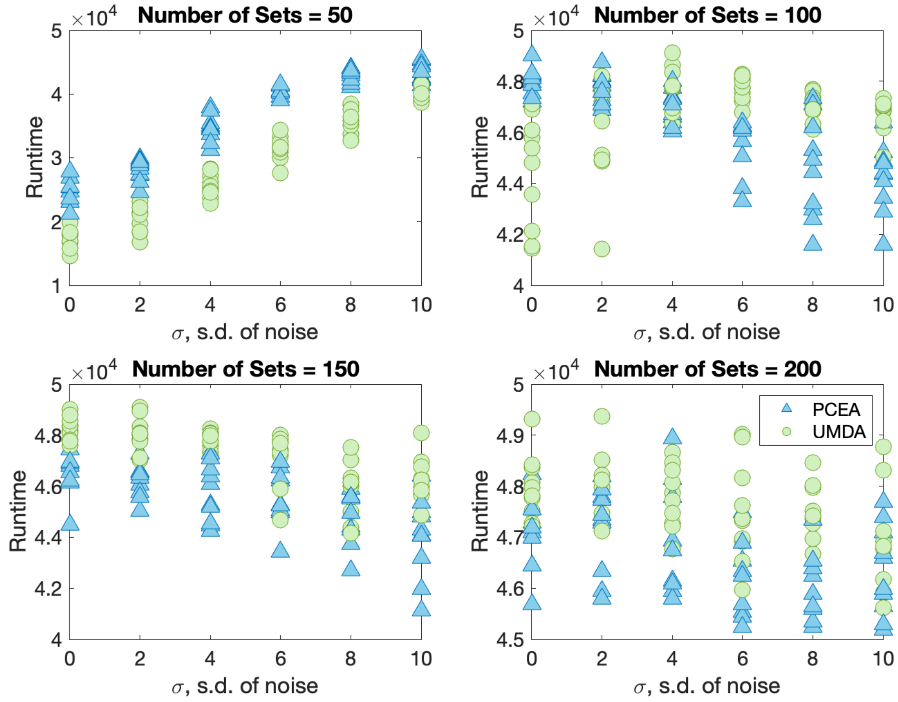

Figure 4.8: Runtime of UMDA (circles) and PCEA (triangles) for best solution found while solving NoisyPenaltySetCover

For both versions of the noisy SetCover problem, a range of 40 problem instances (10 for each problem size) are run with 100 elements and 50, 100, 150 and 200 subsets are available to cover those elements. The problems are created by randomly generating subsets, where the probability of including any element in any subset is p for a chosen $\delta = 0.001$. This is set so that the probability of there being cover is large:

$$(1 - (1 - p)^n)^m = 1 - \delta$$

Therefore, we take:

$$p = 1 - (1 - (1 - \delta)^{1/m})^{1/n}$$

All the algorithms are run until 50,000 function evaluations are reached. An average of 30 runs are reported for both of the versions of SetCover problem. Figure 4.6 reports the best feasible solution found in the fixed budget of function evaluations for the ConstrainedSetCover problem. As evident from the figure, neither of the algorithms can handle noise well. The noisy feasibility check significantly worsens the optimum found even for small standard deviations of noise.

The parameters considered for solving the PenaltySetCover are chosen to be the same as the ConstrainedSetCover. For each noisy PenaltySetCover problem, we plot the best feasible solution found so far in the given function evaluation budget and the runtime in Figures 4.7 and 4.8. It is interesting that both the algorithms can solve the noisy instances in a scalable manner, with UMDA typically producing better quality solutions.

4.5 Summary

The PCEA algorithm employing binary tournament selection and uniform crossover handles noise well on the harder noisy combinatorial problems that we have tried in this chapter. Interestingly, UMDA also handles these cases well, with even a slightly better performance than PCEA. This may be due to the fact that UMDA has a strong selection method (truncation selection) than PCEA (which uses a tournament on pairs of offspring). Of course, parameter values on each could be tweaked to produce slightly different results – our key finding is that these are the only algorithms we have tried that seem remotely practical for such problems. It seems likely that UMDA's resilience to noise is more due to its recombination nature, rather than being considered as an EDA (such as PBIL).

Due to the strong resilience of UMDA on noisy combinatorial problems, we hypothesise that the multi-objective version of UMDA will perform well in multi-objective problems. However, necessary diversification mechanisms may be needed to attain required performance. In the next chapter, we consider a multi-objective formulation of the standard single-objective SetCover problem to understand if the quality of solution may be improved.

Chapter 5

Noisy Multi-objective Combinatorial Problems

A noisy multi-objective combinatorial problem in the search space of binary strings may be defined as follows,

$$f(\mathbf{x}) = (f_1(\mathbf{x}) + N(0, \sigma), f_2(\mathbf{x}) + N(0, \sigma), \ldots, f_k(\mathbf{x}) + N(0, \sigma))$$

where, $\mathbf{x} \in \{0, 1\}^n$ is a candidate solution. The objectives $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})$ are conflicting in nature, so there does not necessarily exist an optimal solution that will minimise all the objectives simultaneously. Instead, there exists a set of non-dominating solutions known as the *Pareto optimal solution set* where none of the objectives may be improved without worsening at least one of the other objectives. In the context of noisy multi-objective optimisation, the goal is to find the set of Pareto optimal solutions, as defined in the absence of noise — however, the challenge is that each time a fitness evaluation is made, noise is applied. This is particularly problematic for algorithms that make use of an *archive* of non-dominated solutions, as it is easy for a solution to be incorrectly placed in the archive due to the noise.

Noise can be detrimental in proper assessment of dominance relationships between solutions in multi-objective optimisation. In the scenario where the solutions are non-dominated, noise can affect the decision of selecting a better solution (in terms of diversity preservation). In this chapter, we are interested to understand which algorithms may be beneficial in solving noisy combinatorial multi-objective optimisation and we empirically examine the performances of several algorithms. We will study the performance of archiving and diversity mechanisms in presence of noise. Here, we focus on discrete problems with additive (posterior) Gaussian noise.

However, most of the previous work on noisy multi-objective optimisation has concerned continuous problems - refer to [53, 109, 41, 40] for some recent studies on noisy multi-objective evolutionary optimisation and [54] for a survey. In order to improve the selection mechanisms in presence of noise for approximating the Pareto front, probabilistic dominance relationships and probabilistic Pareto-ranking have been studied in [113, 61]. The basic idea behind these techniques ensure that, only the solutions that have a high probability of dominance are preferred in instances such as adding to an archive of non-dominated solutions, as well as rejecting solutions that have a high probability of being dominated, like being discarded from the archive. The probabilistic ranking scheme aims to prefer well-spaced solutions (measured in terms of crowding distance) that are identified on the basis of estimated noise variance of every solution with sampling. Resampling methods have been used to calculate expected fitness values for solutions in presence of noise [8, 19, 86]. Instead of resampling all the solutions, a technique based on dominance-dependent lifetime is used to limit the impact of noisy solutions [18]. A short lifetime is assigned to a newly generated solution if the fraction of previously archived solutions that it dominates is large. The solutions whose lifetime has expired are re-evaluated and added to the population. For noise handling, performance indicators have also been used to guide the search process [10].

5.1 Hypervolume Performance Indicator

The *hypervolume* performance indicator [125] is frequently used to assess multiobjective optimisation algorithms. It measures the spread of a set of non-dominated solutions, their closeness to the true Pareto front and provides an assessment if the true Pareto optimal set has been successfully approximated. Where we seek to minimise each objective, this is a measure of the area (or volume) of the region bounded below by a set of candidate solutions simultaneously and bounded above by a reference point r in the objective space.

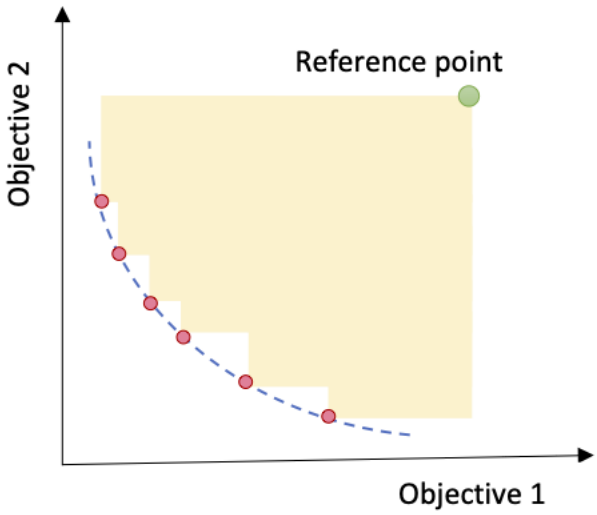

Figure 5.1: Hypervolume performance indicator

The reference point r is chosen to be the maximum value each objective function can attain in each corresponding dimension of the objective space, i.e., $r = (\max f_1, \max f_2, \ldots, \max f_k)$, so that the reference point is dominated by all non-dominated solutions and consequently, each non-dominated solution set has a positive hypervolume contribution. Conversely, for maximisation problems, we take the volume between the candidate set and a lower bounding reference point (in the case of non-negative objectives, it is common to take the origin as the reference point)— please refer Figure 5.1. We use hypervolume of the population as an indicator of the spread of the non-dominated solutions in each generation of the considered algorithms.

Several different algorithms exist for calculation of the hypervolume of a non-dominated set. Here, we have considered an algorithm based on dimension-sweeping for calculating the hypervolume [117, 44]. The hypervolume of a non-dominated point set is calculated by taking the sum of k hypervolumes formed by sweeping a $(k-1)$ dimensional hyperplane through the point set along the first coordinate. Each of the k individual hypervolumes are found by multiplication of the height and the $(k-1)$-dimensional base. In a two-dimensional case, it means that sum of areas of the rectangles shown in yellow in Figure 5.1 is the hypervolume of the non-dominated set of red points.

5.2 Multi-objective Problems Considered

In this chapter, we have studied two noisy multi-objective problems. Initially, we analyse the performance of a collection of multi-objective algorithms on a toy multi-objective problem *Counting Ones Counting Zeroes* (COCZ) (also studied in [69, 67, 17, 95, 33]) without and with high levels of noise and we attempted to identify which algorithms perform better. In the COCZ problem, the first objective function counts the number of ones in a string, and the second objective function counts the number of ones in the first m bits and the number of zeroes in the remainder. We seek to maximise both objectives.

$$\textsc{NoisyCOCZ}(x) =$$

$$\left(\sum_{i=1}^{n} x_i + N(0,\sigma), \sum_{i=1}^{m} x_i + \sum_{i=m+1}^{n} (1-x_i) + N(0,\sigma)\right)$$

The Pareto optimal front consists of strings of the form $1^m *^{(n-m)}$. We compare our algorithms on the basis of the hypervolume performance indicator in a reasonable time budget. The hypervolume of the non-noisy population is considered to estimate how well the algorithms can optimise in presence of noisy function evaluation. It is clearly evident that the liberty of calculation hypervolume of non-noisy populations is not possible in case of realistic problems.

Then the better performing algorithms are chosen to solve the second problem, which is a multi-objective version of the SetCover problem, with the objective function and the constraint as defined in ConstrainedSetCover (defined in Chapter 4) as the two objective functions. These objectives are conflicting in nature. The first objective minimizes the number of sets required to cover all the m elements of the target set, and the second objective minimizes the number of uncovered elements. The noisy version of the multi-objective SetCover problem is defined as follows,

$$\textsc{NoisyMulti-objectiveSetCover}(x) = \left(\sum_{j=1}^{n} x_j + N(0,\sigma), \sum_{i} \left[\sum_{j=1}^{n} a_{ij} x_j = 0 \right] + N(0,\sigma) \right)$$

5.3 Algorithms Chosen for Noisy Multi-Objective Combinatorial Problems

5.3.1 The Simple Evolutionary Multi-objective Optimiser (SEMO)

SEMO [69] is one of the simplest evolutionary algorithms designed for multi-objective optimisation in discrete search space. To the best of our knowledge, it has not previously been used to solve noisy problems. SEMO is a simple population-based algorithm using one-bit mutation, and a variable population size (representing the current non-dominated solutions found).

Algorithm 7: SEMO

Initialise solution $\mathbf{x}$ and add to population $\mathcal{P}$;
repeat
    Choose $\mathbf{y}$ from P and mutate a random bit to get **y'**;
    If **y'** is not dominated by any solution in $\mathcal{P}$ and **y'** $\notin \mathcal{P}$, add **y'** to $\mathcal{P}$ and discard all solutions in $\mathcal{P}$ that **y'** dominates;
end

The algorithm starts with adding an initial solution $\mathbf{x} \in \{0,1\}^n$ chosen uniformly at random to the population $\mathcal{P}$. Then a solution $\mathbf{y}$ is chosen randomly from $\mathcal{P}$ and mutated with a one-bit flip to obtain **y'**. If **y'** is dominated by anything in $\mathcal{P}$ it is discarded. Otherwise it is added to $\mathcal{P}$ and all the solutions that **y'** dominates in $\mathcal{P}$ are discarded. Then a new $\mathbf{y}$ is chosen from $\mathcal{P}$ and the process is repeated. One of the great challenges SEMO will face due to noisy dominance relations is that, often good solutions will be discarded and bad solutions will be retained in the population $\mathcal{P}$.

5.3.2 Non-dominated Sorting GA – II (NSGA–II)

The NSGA–II by Deb et al. [27] sorts the population into non-dominated fronts in each generation. Based on non-dominated sorting and using a crowding heuristic to break ties, the best half of individuals become the parent population of the next generation. In case of noisy function evaluations, non-dominated sorting will be affected and worse solutions will appear in better non-dominated fronts. The algorithm structure is given in Algorithm 8.

Algorithm 8: NSGA–II

Initialise random parent population $\mathcal{P}$ of size $\lambda/2$;
Sort $\mathcal{P}$ by non-domination rank ;
Generate offspring population $\mathcal{O}$ of size $\lambda/2$ by binary tournament selection, single point crossover and bit-wise mutation;
repeat
    Combine parent and offspring population, $\mathcal{R} = \mathcal{P} \cup \mathcal{O}$;
    Generate fronts $\mathcal{F}_i$ with non-dominated sorting;
    Set new parent population $\mathcal{P}' = \phi$ and $i = 1$;
    while $|\mathcal{P}'| + |\mathcal{F}_i| \leq \lambda/2$ **do**
        Add $\mathcal{F}_i$ to $\mathcal{P}'$;
        Set $i = i + 1$;
    end
    Choose least crowded solutions from $\mathcal{F}_i$ and add to $\mathcal{P}'$ until $|\mathcal{P}'| = \lambda/2$;
    Generate new offspring population $\mathcal{O}'$ of size $\lambda/2$ by binary tournament selection, single point crossover and bit-wise mutation;
end

5.3.3 Variants of Multi-objective Univariate Marginal Distribution Algorithm (moUMDA)

From our experiments in noisy single-objective combinatorial problems, UMDA and PCEA show significantly better performance in handling noise compared to the other algorithms we tried, with UMDA generally producing better quality solutions. From these results, we hypothesise that a multi-objective version of UMDA (denoted moUMDA) may be able to handle large levels of noise in noisy combinatorial multi-objective problems if proper diversification mechanisms are employed. In order to investigate this, we have considered several versions of moUMDA in our analysis with different diversification techniques.

Pelikan et al. [88] introduced a version of UMDA to address multi-objective problems which used non-dominated sorting in the selection mechanism. They also experimented with clustering methods, to help the algorithm generate solutions across the Pareto front. We have followed this idea, and studied several versions of UMDA adapted for multi-objective problems. Where non-dominated sorting and crowding are used for selection, these are implemented identically to NSGA–II. We also consider making use of an archive, and in using hypervolume as a criterion in selection:

moUMDA without duplicates Uses non-dominated sorting (with crowding to break ties) for selection. Maintains diversity by disallowing duplicates when generating the population. See Algorithm 10.

moUMDA with clustering Uses non-dominated sorting (with crowding to break ties) for selection. Clusters the selected population members (using either K-means or Hierarchical Agglomeration), and produces a frequency vector for each cluster. Generates next population from these, in proportion to the number of items within each cluster. See Algorithm 11.

moUMDA with Pareto archive Maintains an archive of non-dominated solutions and uses this to generate the frequency vector for the next population. Uses non-dominated sorting (with crowding to break ties) for selection, and updates the archive with the selected items. See Algorithm 12.

moUMDA with hypervolume comparison operator Uses binary tournament selection, comparing solutions initially by Pareto dominance. If neither dominates the other, then select the one with the better hypervolume indicator value. See Algorithm 13.

Algorithm 9: moUMDA

Initialise frequency vector $p = (0.5, \ldots, 0.5)$;
repeat
    Generate population of size λ from p;
    Use non-dominated sorting and crowding to select the best μ individuals;
    Update frequency vector p based on selected individuals;
end

Algorithm 10: moUMDA without duplicates

Initialise frequency vector $p = (0.5, \ldots, 0.5)$;
repeat
    Generate population of size λ from p, disallowing duplicates;
    Use non-dominated sorting and crowding to select the best μ individuals;
    Update frequency vector p based on selected individuals;
end

Algorithm 11: moUMDA with clustering

Set $k = \lfloor \sqrt{\mu} \rfloor$ as the number of clusters;
Initialise frequency vectors $p_i = (0.5, \ldots, 0.5)$ for each $i = 1 \ldots k$;
Set $q_i = \mu/k$ for each $i = 1 \ldots k$;
repeat
    Generate population of size $2q_i$ from p_i, for each $i = 1 \ldots k$;
    Use non-dominated sorting and crowding to select the best μ individuals from all the populations;
    Cluster the selected individuals into k clusters;
    Let q_i be the number of individuals in cluster i, for each $i = 1 \ldots k$. Update frequency vectors p_i based on selected individuals in each cluster;
end

Algorithm 12: moUMDA with Pareto archive

Initialise frequency vector $p = (0.5, \ldots, 0.5)$;
Initialise empty archive $\mathcal{A}$;
repeat
    Generate population of size λ from p;
    Use non-dominated sorting and crowding to select the best μ individuals;
    Add these to archive $\mathcal{A}$ and remove any dominated solutions;
    Update frequency vector p based on archive $\mathcal{A}$;
end

Algorithm 13: moUMDA with hypervolume comparison

```
Initialise frequency vector p = (0.5, ..., 0.5);
repeat
    Create empty population P;
    repeat μ times
        Generate two strings, x and y from p ;
        Add string with best hypervolume to P
    end
    Update frequency vector p based on population P;
end
```

5.4 Experiments — Noisy Multi-objective Problems

Following the same strategy as for single objective problems, we initially choose a wide range of evolutionary multi-objective algorithms to compare their performances on a toy problem: noisy CountingOnesCountingZeroes (COCZ). The algorithms considered for solving COCZ consist of SEMO, NSGA–II and several versions of multi-objective UMDA (moUMDA) as described above. Depending on their performances on this problem, we selected a smaller set of the better performing algorithms for the multi-objective noisy SetCover problem.

Some recent studies claim that multi-objective evolutionary approaches are useful in solving single objective optimisation problems [105]. For example, the multi-objective version of SetCover could enable us to find good solutions to the original single-objective version (by looking at solutions generated which do not violate the constraints). Here in subsection 5.4.2, we consider whether this approach is also helpful in the context of noise.

5.4.1 Noisy CountingOnesCountingZeroes (COCZ)

In this subsection, we solve a toy multi-objective problem, the noisy COCZ with two problem sizes. The first one with $n = 20, m = 10$ and with additive Gaussian noise centered at zero and having standard deviations $\sigma = 01, 3, 5, 7$ and 9. And the second problem with $n = 30, m = 15$ and with additive Gaussian noise centered at zero and having standard deviations $\sigma = 0, 1, 3, 5, 7, 9, 11, 13$ and 15. We set the parameter $\mu = \lambda/2$, where $\lambda = 20\sqrt{n}\log n$ for all the versions of moUMDA. For NSGAII, the parent population size is set as $10\sqrt{n}\log n$.

All the algorithms are run for 50,000 function evaluations and the mean of 30 runs are reported. The best hypervolume of the population found so far in the fixed budget of function evaluations are reported in Figures 5.2 and 5.3. For the smaller problem size, the Pareto optimal front would contain 2^{10} elements and the best possible hypervolume would be 345. For the larger problem size, the Pareto optimal front would contain 2^{15} elements and the best possible hypervolume is 780. We have used the dimension-sweep algorithm for calculating hypervolume and the source code by [44] for hypervolume calculation in the experiments.

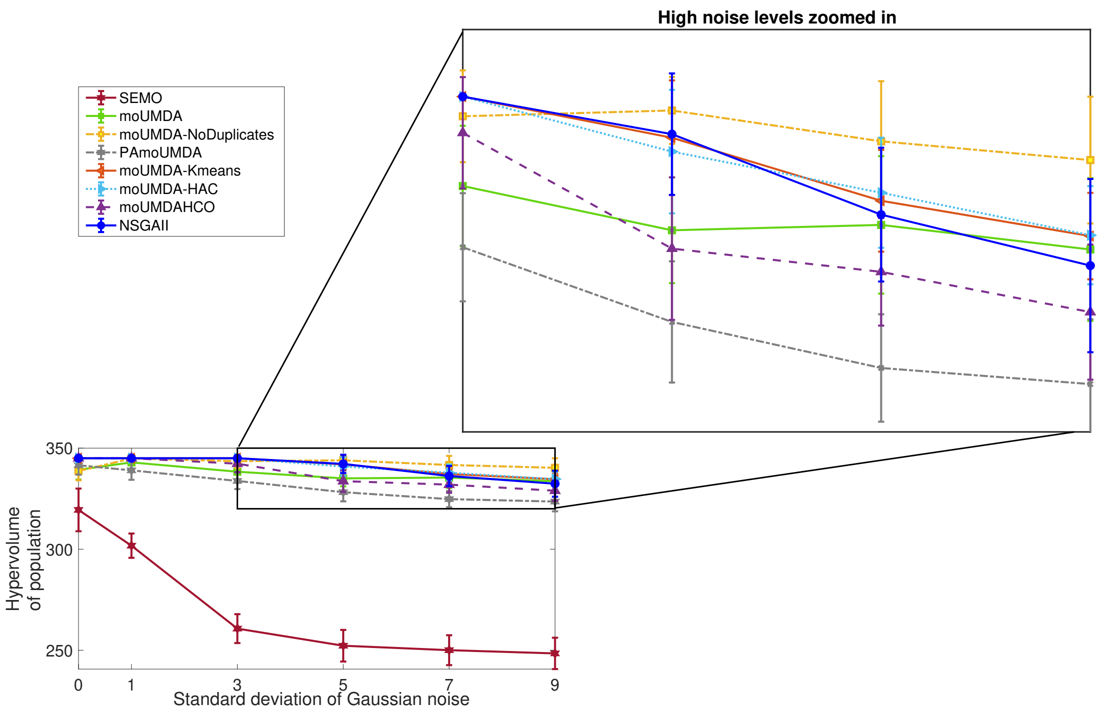

Figure 5.2: Comparison of the hypervolume of population while solving the noisy COCZ with $n = 20, m = 10$

The results shown in Figures 5.2 and 5.3 show that SEMO is the worst performing algorithm, even when there is no noise, and the performance degrades slightly as noise is increased. The Pareto Archive algorithm (PAmoUMDA) is the next worst. Although it does not degrade too much with the added noise, it is still clearly worse than the other algorithms. It is evident that due to noisy fitness evaluations, dominated solutions get added to the archive frequently. Consequently, widely studied approaches like Strength Pareto Evolutionary Algorithm (SPEA) where the archive is clustered to maintain a given size of the archive, might not be beneficial in noisy combinatorial optimisation with large standard deviations of noise in fitness evaluations.

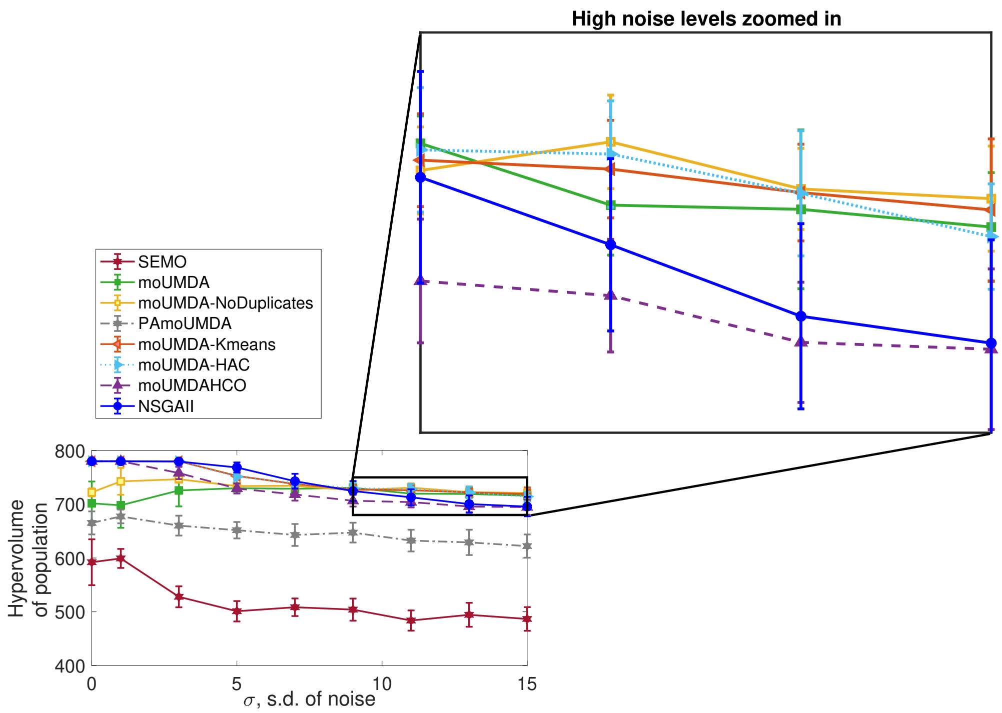

Figure 5.3: Comparison of the hypervolume of population while solving the noisy COCZ with $n = 30, m = 15$

The remaining algorithms have rather similar performance, but we can still distinguish different behaviours by looking at the zoomed in section of the plot in Figures 5.2 and 5.3. The version of moUMDA that uses the hypervolume comparison operator (moUMDAHCO) performs very well when there is little or no noise. In presence of noisy fitness evaluations, the hypervolume perfomance indicator based search misjudges which string would provide a better hypervolume to the population. Hence, its performance degrades considerably as the level of noise increases. The same is true for NSGA–II. When the noise reaches a standard deviation of $\sigma = 15$ for the problem with $n = 30$, these two algorithms are the worst of the remaining ones.

The plain moUMDA and the version forbidding duplicates in population both have a curious property that their performance improves with the presence of low levels of noise, and then degrades at higher levels of noise. In presence of noisy fitness evaluations, the same strings might have different fitness values and as a result, duplicates will get included in the population. As the standard deviation of noisy fitness evaluation increases, even though the process of removing duplicates is carried out, the chance of having duplicates in the population increases. When there is no noise in fitness evaluation, although removing duplicates ensure diversity, it is done at the cost of additional fitness evaluations. We speculate that low levels of noise allow for more diversity in the populations but the additional fitness evaluations required to remove duplicates is not as high as in the case when there no noise. At the highest levels of noise, the process of removing duplicates is not as effective as in case of non-noisy or noisy fitness evaluations with small standard deviation.

At high levels of noise ($\sigma = 15$) the plain moUMDA and the version forbidding duplicates are the best performing algorithms, along with the two versions of moUMDA that use clustering (moUMDA-Kmeans and moUMDA-HAC). moUMDA with no duplicates is marginally the best overall at this level of noise.

5.4.2 Noisy Multi-objective SetCover

In this section, we compare the performance of three of our multi-objective algorithms, viz., NSGA–II, moUMDA with no duplicates allowed and moUMDA employing K-means clustering, on the noisy multi-objective SetCover problem. We have chosen these algorithms based on their behaviours on the COCZ. These were amongst the best algorithms we tried on that problem. There being little to distinguish the two different clustering methods, we have chosen to test just one of these (K-means clustering). We have selected the "no duplicates" version of moUMDA, as this gave a small advantage over the plain moUMDA. And we have kept NSGAII as this is a standard algorithm for any multi-objective problem.

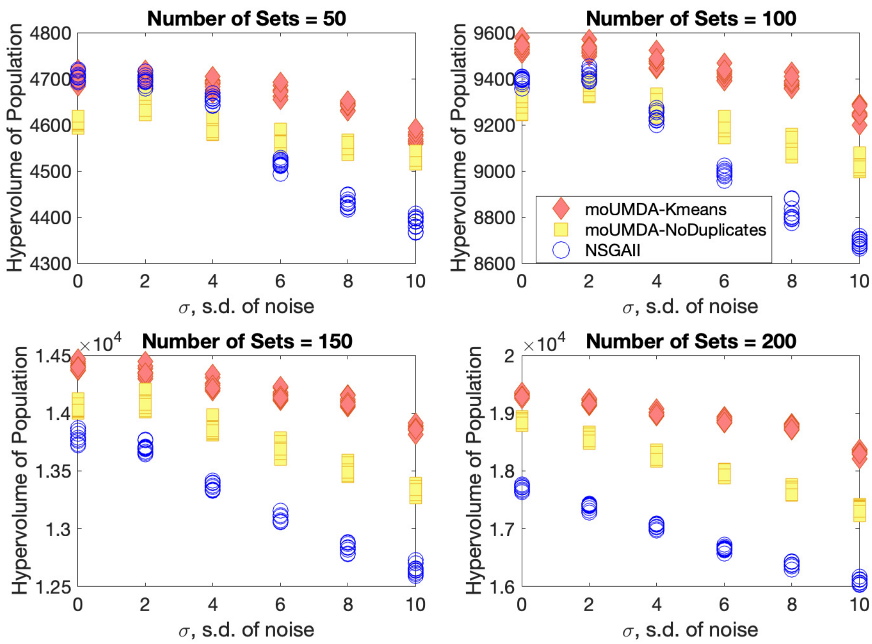

Figure 5.4: Best hypervolume of population obtained for Multi-objective SetCover

We have considered problem sizes $50, 100, 150$ and 200. Corresponding to each of the four problem sizes, 10 different random problem instances (same as in the single-objective SetCover problem considered in Chapter 4) are considered, i.e., a total of 40 different problem instances have been studied.

All the algorithms are run for 50,000 function evaluations. The best hypervolume of the population obtained in the fixed function evaluation budget for each of 30 runs is shown in Figure 5.4. We observe that the clustering algorithm, moUMDA-Kmeans, handles high levels of noise significantly better than other algorithms. It is evident that, the performance of NSGA–II becomes worse as the standard deviation of noise increases and the problem size increases and indeed is the worst of the three algorithms on this problem.

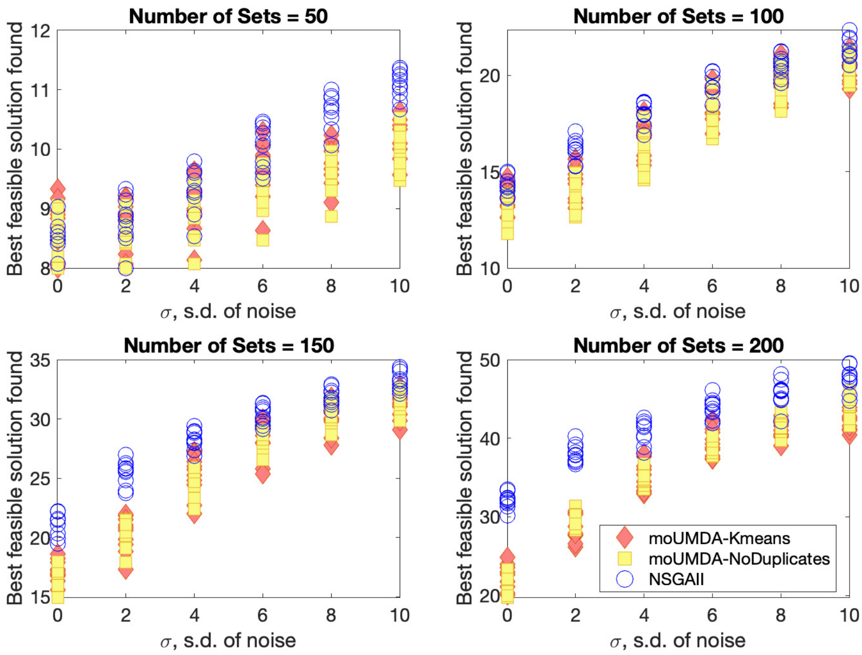

Figure 5.5: Best feasible solution found while solving the noisy Multi-objective SetCover

We also consider the multi-objective formulation of noisy SetCover as a means to solving the standard single objective problem. To this end, we consider the quality of the best feasible solutions found by each algorithm, averaged over the 30 runs. The results are plotted in Figure 5.5. Again, the two versions of moUMDA perform better than NSGA–II. A comparison with Figure 4.7 shows that this approach can indeed produce better quality results than the single objective formulation.

5.5 Summary

We are not aware of previously published results on noisy combinatorial multi-objective problems. We carefully selected a set of multi-objective algorithms on the basis of the performance on noisy COCZ and tested them on the noisy multi-objective SetCover. We observe that multi-objective UMDA with a simple diversity mechanism that allows no duplicate solutions in the population is effective at solving the noisy SetCover problem in both constrained and multi-objective forms. UMDA can also benefit from using a clustering approach when dealing with noisy multi-objective problems.

Chapter 6

UAV Path Planning — A Realistic Noisy Multi-objective Combinatorial Problem

Unmanned Aerial Vehicles (UAVs) have now become essential in a multitude of surveillance and accessibility tasks in several domains. The path of the UAV needs to be planned, to have maximum coverage of the area under surveillance. An important aspect of this optimisation problem consists of identification of the optimum path through specific coordinates in the air, in order to obtain the maximum ground visibility, by keeping in mind the resource constraints.

A significant challenge in UAV surveillance is posed by occlusions in the visibility region of the UAV due to the presence of uncertain weather factors, complex urban landscapes, or dense mountainous terrains. A realistic scenario of UAV path planning is considered in this chapter, where, due to uncertain weather factors such as clouds, fog and haze, the visibility of the UAV is obscured with a random noise.

In this chapter, the UAV path planning scenario in the presence of weather factors is formulated as a combinatorial multi-objective problem affected by noise. Solution methodologies of the problem with multi-objective evolutionary algorithms (MOEAs) are proposed.

In the literature, the placement of aerial drones in a certain range above uneven terrain has been modelled as a 3D Art Gallery problem [104], which is closely related to the SetCover problem. The sweep algorithm [107], the polynomial-time greedy algorithm [106] and the 3–coloring method [123] have been implemented to identify the optimal coverage set. However, reduced visibility due to weather factors, variable illumination etc. has not been considered in these models. In this chapter, the vertices of the UAV path are selected and optimized in a similar fashion like the SetCover problem, however the visibility takes into consideration the path taken by the UAV and the presence on uncertain weather influences within the path.

The chapter is organised as follows. The next section illustrates the mathematical formulation of the UAV surveillance scenario as a noisy multi-objective combinatorial problem. Due to the promising results shown by multi-objective versions of UMDA (moUMDA) in noisy multi-objective combinatorial problems, moUMDA is carefully adapted to this problem scenario. In a recent study [115], a surveillance mission for mutliple aerial vehicles are planned with an adapted version of Non-dominated Sorting GA-II (NSGA–II). The performance of our proposed algorithms is compared with a version of NSGA–II adapted to our noisy combinatorial multi-objective problem. The algorithm structures are described in the third section and the experimental results are discussed in the fourth section.

6.1 Problem Description

For the surveillance of the ground surface G, a UAV is to be deployed to follow an optimal path to maximise the surveillance region. The air surface above G is considered as a combination of parallel grids-like structures of the same dimension as G at different altitudes. The locus of the UAV may be considered as a graph with vertices across the air grids. The UAV has a pre-specified visibility that can be assumed as a conical structure whose apex is located on the vertices of the graph, such that from each grid point in air, a circular region of ground is visible (refer Figure 6.1). The ground surface can be considered as a composition of the circular bases of the visibility cones.

The UAV has an angle of observation θ which is the apex angle of visibility cone. The UAV may be present at any altitude in between h_{min} and h_{max}. Let Q be a grid-like structure at height h and E be the projection of air grid on the ground having $p_1 \times p_2$ grid points. The UAV may be placed at some point $(q_a, q_b, h) \in Q$ such that $h \in [h_{min}, h_{max}]$, and the projection of the location of UAV on E is $(e_a, e_b) \in E$. Thus, Q and E may be written as follows.

$$\begin{aligned} Q &= \{(q_a, q_b, h), \text{ such that } h_{min} \leq h \leq h_{max} \\ &\quad \text{and } a = 1, \ldots, p_1, b = 1, \ldots, p_2\} \quad (6.1) \\ E &= \{(e_a, e_b), \text{ such that} \\ &\quad a = 1, \ldots, p_1, b = 1, \ldots, p_2\} \quad (6.2) \end{aligned}$$

The vertices of the path of the UAV may be located on any of the air grid points, say, on $n = p_1 \times p_2 \times |h_{max} - h_{min}|$ positions. Now, we consider n decision variables x_j indicating the choice of an air grid point as a vertex in the path of the UAV, defined in the following manner.

$$x_j = \begin{cases} 1, & \text{if UAV position } j \text{ in air grid Q considered} \\ 0, & \text{otherwise,} \qquad\qquad \forall j = 1, \ldots, n \end{cases} \quad (6.3)$$

Let ρ be the visibility radius of the UAV. Let the collection of the visible points

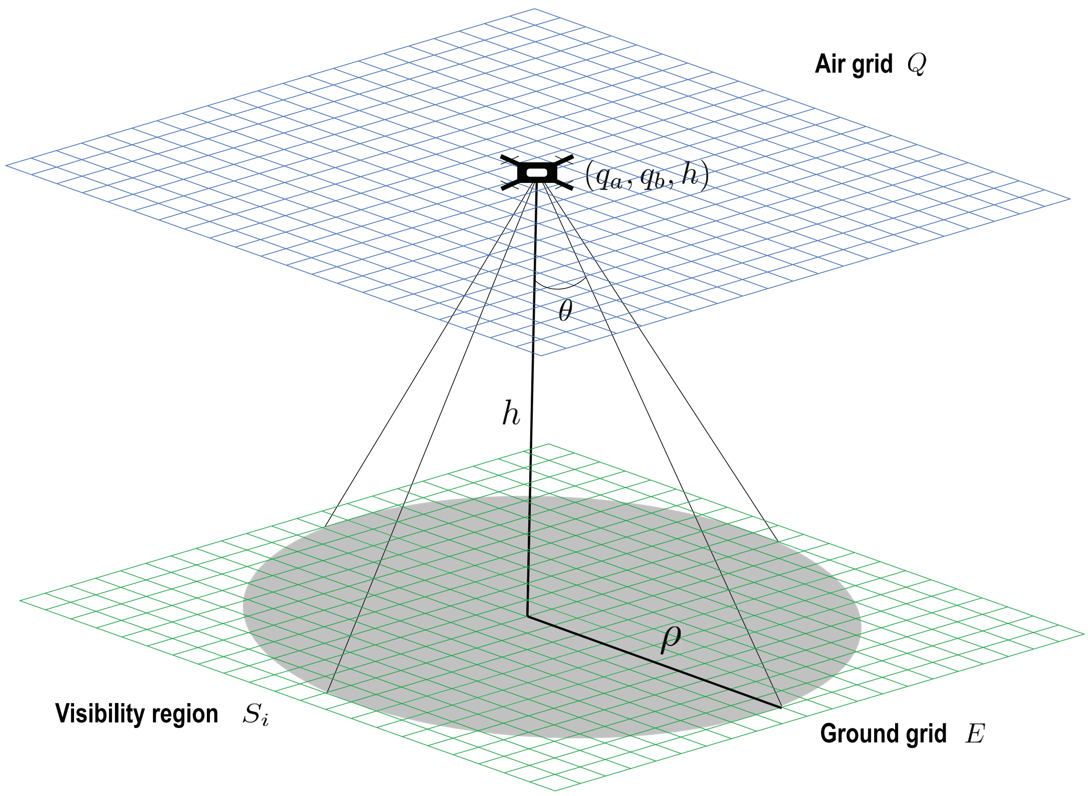

Figure 6.1: Visibility region of a UAV

in E from j^{th} UAV location be S_i, $\forall i \in \{1, 2, \ldots, (p_1 \times p_2)\}$ where, $S_i \subseteq E$ and

$$
\begin{aligned}
S_i &= \{(s_a, s_b) \in E, \text{ such that} \\
&\quad (e_a, e_b) \text{ is the } i^{th} \text{ point of } E\}
\end{aligned}
\tag{6.4}
$$

The circular visibility base with radius ρ of the visibility cone is defined as follows,

$$
\sqrt{(s_a - e_a)^2 + (s_b - e_b)^2} \leq \rho \tag{6.5}
$$

Here, the apex angle θ depends on the sensor configuration of the UAV. The visibility cone for each UAV depends on the altitude h as follows,

$$
\rho = h \tan\left(\frac{\theta}{2}\right), \quad h_{min} \leq h \leq h_{max} \tag{6.6}
$$

In order to formulate the objectives for path planning optimization problem let us consider that the UAV moved from an arbitrary point $x_u = (q_{a_u}, q_{b_u}, h_u)$ to $x_v = (q_{a_v}, q_{b_v}, h_v)$ and will move to $x_w = (q_{a_w}, q_{b_w}, h_w)$, where $x_u, x_v, x_w \in Q$. It is to be noted that, UAV can move from x_v to x_w only if the movement satisfies the specific turning radius of UAV, which means $cos\angle x_u x_v x_w$ must be in between predefined threshold ϵ_1 and ϵ_2, where,

$$cos\angle x_u x_v x_w = \frac{\overrightarrow{x_v x_u} \cdot \overrightarrow{x_v x_w}}{\|\overrightarrow{x_v x_u}\| \|\overrightarrow{x_v x_w}\|} \tag{6.7}$$

Therefore, if $d(x_v, x_w)$ is the Euclidean distance between x_v and x_w, then it is required to minimize the total distance covered by the UAV provided that the turning radius is in the predefined threshold. Then the first objective in the optimization problem is defined as follows:

Objective 1:

$$\text{Minimize distance covered} = \sum_{v=1}^{n} \sum_{w=1}^{n} d(x_v, x_w), \tag{6.8}$$

$$\text{subject to, } \epsilon_1 < cos\angle x_u x_v x_w < \epsilon_2$$

For the movement of the UAV through n air grid points, the visibility of the ground region by the UAV is calculated from the vertices in its path. We are interested to find an optimal set of the visible regions S_is such that maximum area under surveillance at the ground level is covered. The *incidence matrix* $A = (a_{ij})$ is a Boolean matrix which represents the visibility coverage of the projected grid points in the ground such that

$$a_{ij} = \begin{cases} 1, & \text{if } j^{th} \text{ UAV location can cover } S_i \\ 0, & \text{otherwise} \end{cases} \tag{6.9}$$

In case of the second objective, we are interested to obtain the number of points in the ground surface which are not visible from the chosen set of vertices in the UAV path. Here, $\sum_{j=1}^{n} a_{ij}x_j, \ \forall i$, counts the number ground points visible from the chosen set of air grid points. Subtracting the summation from an all ones string with $p_1 \times p_2$ ones would give the number of uncovered points. Here, $1_{p_1 \times p_2}$ is a column matrix with $(p_1 \times p_2) \times 1$ rows of 1s. It is to be noted that, since $a_{ij}, \ \forall i$ and $x_j, \ \forall j$ are binary strings, the summations over j are binary operations.

Objective 2:

$$\text{Minimize uncovered points by chosen } S_i\text{s} = \sum_{i=1}^{p_1 \times p_2} \left(1_{p_1 \times p_2} - \sum_{j=1}^{n} a_{ij}x_j \right) \tag{6.10}$$

This combinatorial problem with two objectives, targets minimising the distance covered by the UAV as well as minimising the uncovered points in the projected grid on the ground simultaneously. These two objectives are competitive to each other, because increasing the distance coverage of the UAV ensures a higher possibility of coverage of the ground. Hence there exists a trade-off between these two objectives. That is, achieving the best for one objective leads to the compromise of the other objective functional value. So, our aim is to find a set of Pareto optimal (or non-dominated or efficient) choices for this problem. Here, the problem is formulated as a multi-objective combinatorial optimization problem for which a Pareto optimal set needs to be obtained. A number of constraints may be considered in this optimization setup, for example, a pre-specified percentage of total ground coverage may be considered if there is a requirement. Uncertainty may arise in this optimization set-up in presence of weather factors such as clouds, haze and fog. From each vertex of the UAV path, the whole circular region of the ground might not be visible in that case. Please refer Figure 6.2 which illustrates the change in visibility region due to clouds. The visibility circle will be obscured

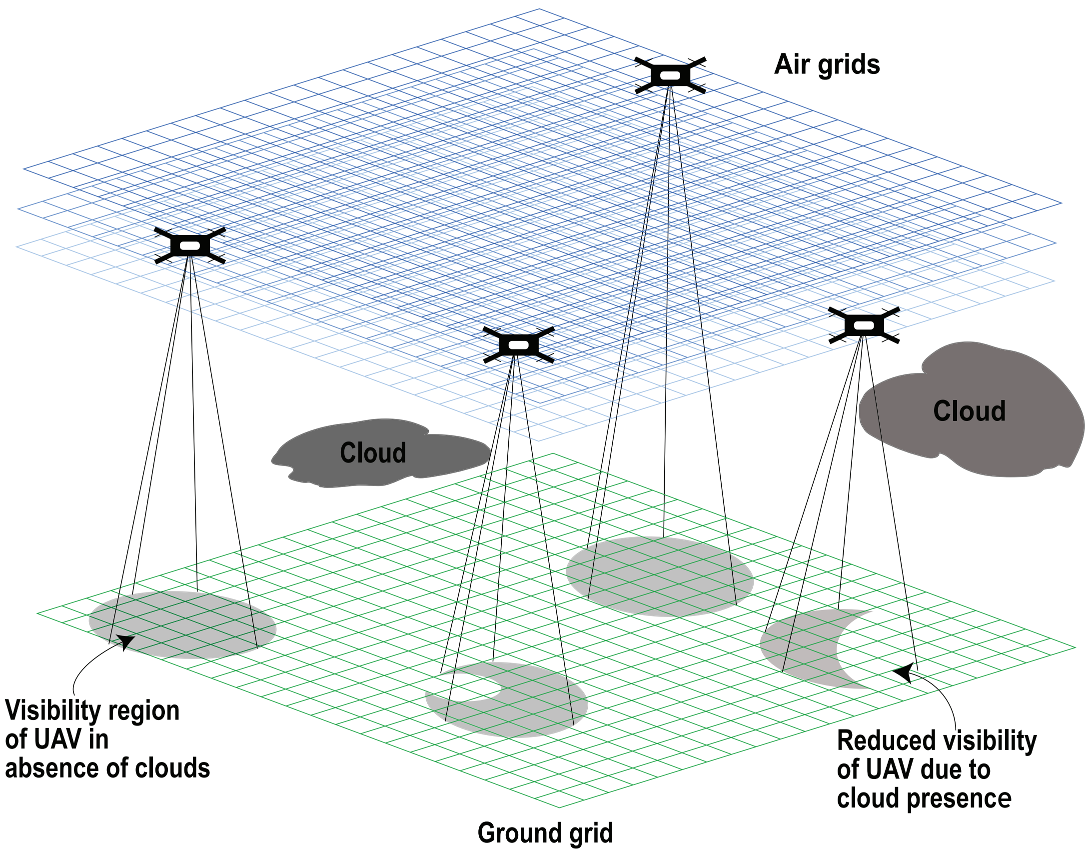

Figure 6.2: Reduced visibility of UAVs due to presence of clouds

and the ground points belonging in it may be different at each time instance based on a stochastic factor.

Here, a $p_1 \times p_2 \times |h_{max} - h_{min}|$ cloud map C^{map} is considered which is known beforehand from the weather report, location survey map etc. In the cloud map, the presence of 1 at a grid point denotes the possibility of cloud at that location, then, the corresponding point in the ground has a chance to get occluded. Then each S_i corresponding to $(q_a, q_b, h) \in Q$ might cover a different subset of elements in E for each time instance, than in the ideal condition.

$$C^{map}_{(q_a,q_b,h)} = \begin{cases} 1, & \text{cloud present at } (q_a, q_b, h) \in Q \\ 0, & \text{otherwise} \end{cases} \tag{6.11}$$

The radius and/or the centre of cloud coverage in the visibility circle may also get perturbed due to weather factors. For the sake of simplicity, we have considered that each visibility circle may contain at most one cloud-dependent occlusion $O_i, \forall i$ which is circular in nature with a noisy radius. However, several disjoint/overlapping occlusions of different sizes may exist in reality, which may be considered in the model in future. The occluded region is considered to be a noisy circle with a radius μ with an additive Gaussian noise with mean zero and a standard deviation dependent on the weather report. Let the centre be $(\varepsilon_a, \varepsilon_b) \in E$ of the occluded region in the ground surface be $O_i = \{(o_a, o_b) \in E\}$ corresponding to S_i. Here,

$$\sqrt{(o_a - \varepsilon_a)^2 + (o_b - \varepsilon_b)^2} \leq \mu + N(0, \sigma) \tag{6.12}$$

The noisy visibility region $S_i^{noisy} \quad \forall i \in \{1, ..., (p_1 \times p_2)$ then becomes

$$S_i^{noisy} = \begin{cases} S_i \backslash O_i, & \text{if } C^{map}_{(q_a, q_b, h)} = 1 \\ S_i, & \text{otherwise} \end{cases} \tag{6.13}$$

Here, the noisy visibility region S_i^{noisy} is considered as the difference between the visibility region and the occlusion region. With the inclusion of the noise parameter, the problem will be significantly harder to solve as it will interfere with the visibility region evaluations. Following section illustrates the algorithms to solve the stated multi-objective combinatorial problem in noisy environment.

6.2 Algorithm Description

Much of the previous work in solving multi-objective combinatorial optimisation in noisy environment concerns continuous problems. In Chapter 4, the noisy versions of single-objective un-constrained and constrained combinatorial optimisation problems, such as SubsetSum, Knapsack and SetCover with large noise variance, have been solved reasonably with an algorithm employing only selection and uniform crossover, PCEA and an estimation of distribution algorithm, UMDA, with UMDA having a slightly better performance than PCEA.

Algorithm 14: moUMDA for path planning of UAV in presence of noisy visibility

```
Initialise generation counter t = 0;
Initialise random population P_Q(t) of vertices of UAV path on air grid Q;
Find shortest path (Objective 1) w.r.t. turning radius for each p ∈ P_Q ;
Calculate uncovered points w.r.t. S_i^noisy ∀i for each p ∈ P_Q along the
  shortest path with Bresenham Algorithm (Objective 2);
while Termination condition reached do
    Sort population into fronts, NoisyNDSorting(P_Q) = (F_1, F_2, ...);
    Choose best half of population with NonDominatedRank and
      CrowdingDistance;
    Initialise the next generation population P_Q(t + 1) = Φ and the front
      counter i = 1;
    while |P_Q(t + 1)| + |F_i| ≤ λ/2 do
        Include F_i) in next population, P_Q(t + 1) ∪ F_i = P_Q(t + 1);
        Check next front for inclusion i = i + 1;
    end
    Calculate CrowdingDistance(F_i);
    Sort F_i with respect to crowding distances;
    Include remaining individuals to P_Q(t + 1), such that
      P_Q(t + 1) = P_Q(t + 1 ∪ F_i[1 : (λ/2 − |P_Q(t + 1)|)];
    Increment t = t + 1;
    Generate population distribution matrix D;
    Choose next generation of population randomly w.r.t. D;
end
```

The efficiency of these algorithms in handling noise may be attributed to several reasons. Both of these algorithms are population-based and it has been recognised for a long time that the population size is an important factor to consider while analysing the ability of an EA to handle noise [55, 98]. Interestingly, the probabilistic model building mechanism of the UMDA has a resemblance to *genepool crossover*, where at each bit position, the offspring bit is obtained by recombination of that bit across the whole parent population. It also seems likely that UMDA's better performance may be attributed to its stronger selection method (truncation selection) than PCEA (which uses a tournament on pairs of offspring).

Algorithm 15: NoisyNDSorting($\mathcal{P}_Q$)

Result: Non-dominated fronts $\mathcal{F}$ of $\mathcal{P}_Q$
for *each individual* $p \in \mathcal{P}_Q$ **do**
  Initialise set of solutions dominated by p , $S_p = \Phi$;
  Initialise number of solutions that dominate p, $n_p = 0$;
  for *each individual* $q \in \mathcal{P}_Q$ **do**
    if *if* p *dominates* q *noisily i.e,.* $p \prec q$ **then**
      Add q to the set of solutions dominated by p, $S_p = S_p \cup \{q\}$;
    else if q *dominates* p *noisily,* $(q \prec p)$ **then**
      Increment domination counter of p, $n_p = n_p + 1$;
  end
  if p *belongs to the first front, i.e.,* $n_p = 0$ **then**
    $\mathcal{F}_1 = \mathcal{F}_1 \cup \{p\}$;
  end
end
Initialise the front counter, $i = 1$;
while $\mathcal{F}_i \neq \Phi$ **do**
  Set $Q = \Phi$, which is used to store the members of the next front;
  for $p \in \mathcal{F}_i$ **do**
    for $q \in S_p$ **do**
      Decrement the number of solutions dominating q, $n_q = n_q - 1$;
      if q *belongs to the next front,* $n_q = 0$ **then**
        $Q = Q \cup \{q\}$;
      end
    end
  end
  Increment $i = i + 1$;
  Store the next front $\mathcal{F}_i = Q$;
end

Pelikan et al. [88] introduced a version of UMDA to address multi-objective problems which used non-dominated sorting as the selection mechanism. The Chapter 5 investigates similar adaptations of moUMDA using non-dominated sorting and crowding distance to break ties, but several diversification mechanisms that are efficient is solving noisy combinatorial multi-objective problems, in practice. A similar version of moUMDA, is adapted according to the requirements in this UAV path planning problem. The proposed algorithm structure of moUMDA that is considered here is described in Algorithms 14, 15 and 16.

A population $\mathcal{P}_Q$ is chosen uniformly at random where each individual is a $p_1 \times p_2 \times h$ Boolean matrix denoting the vertices of the path of the UAV on the air grids with a one and zero, otherwise. For each population individual, the shortest path between the vertices of the UAV path is calculated by keeping the constraint of turning radius in consideration (with equation 6.7).

For each population individual, the lower leftmost grid point is considered as the first vertex in the UAV path. The nearest neighbouring grid point that has a one as well as that satisfies the turning radius constraint is added to the UAV path as the next vertex. If the turning radius constraint is violated, the next nearest neighbouring grid point that has a one, is checked. A list of visited and un-visited vertices is maintained at each iteration. It may be the case that once the shortest path finding process finishes, some vertices remain un-visited since they violated the turning radius constraint from each of the visited vertices. These un-visited vertices are not added to the UAV path and the population individual is modified accordingly.

Along the calculated shortest path, with the help of Bresenham Algorithm [16], the grids are identified through which the UAV may pass and hence the visibility may be calculated. According to the cloud map C^{map} known beforehand from the weather report, the corresponding noisy visibility region S_i^{noisy} is calculated with the help of equations 6.12 and 6.13.

Now, the population is sorted with respect to non-dominance and each of the individuals are assigned to corresponding non-dominated fronts (please refer to Algorithm 15). However, when the fitness evaluations i.e., the visibility region calculations are noisy, non-dominated sorting will be affected and worse solutions will appear in better non-dominated fronts. The best half of the population individuals are selected (truncation selection mechanism) on the basis of their non-domination

Algorithm 16: CrowdingDistance($\mathcal{F}_i$)

$l = |\mathcal{F}_i|$ number of solutions in $\mathcal{I}$;
for *all idx* **do**
  Initialise distance $\mathcal{F}_i[idx]_{distance} = 0$;
end
for *all objectives m* **do**
  Sort using each objective value $\mathcal{F}_i = sort(\mathcal{F}_i, m)$;
  Set $\mathcal{F}_i[1]_{distance} = \mathcal{F}_i[l]_{distance} = \infty$ so that the boundary points are selected;
  for $i = 2$ $to(l-1)$ **do**
    $\mathcal{F}_i[i]_{distance} = \mathcal{F}_i[i]_{distance} + (\mathcal{F}_i[i+1].m - \mathcal{F}_i[i-1].m)/(f_m^{max} - f_m^{min})$;
  end
end

rank with ties being broken by the crowding distance calculation. The choice of individuals with maximum crowding distance helps to spread the solutions all across the Pareto front. The algorithm for crowding distance calculation is explained in Algorithm 16.

Since the UMDA belongs to the EDA-schema, it learns the probabilistic model of the promising candidate solutions obtained from truncation selection. In this adapted version of moUMDA, the probability distribution matrix $\mathcal{D}$ is generated from the best half of the population individuals by obtaining the frequency of ones in each air grid point. The next generation of population is chosen uniformly at random with respect to the probability distribution matrix $\mathcal{D}$. The procedure of generation of $\mathcal{D}$ and the corresponding generation of population is continued until the termination condition is reached.

A major issue in evolutionary multi-objective optimisation is maintaining the diversity within the solution set. Primarily, breaking the ties with the crowding distance helps the algorithm to promote solutions from the less-crowded areas within the non-dominated solutions. However, this diversity mechanism may not be enough when the feasible region is large. In order to let the candidate solutions be sampled from different regions of the Pareto front, i.e., bring diversity into the population, a clustering mechanism in considered in another version of the moUMDA. K-means clustering is employed (considering the idea in Chapter 5), to divide the best half of population individuals into $k = \sqrt{\lambda}$ clusters, where λ is the population size. From each K-means cluster, a probability distribution matrix $\mathcal{D}_k$ is generated. The number of new candidate solutions sampled from each $\mathcal{D}_k$ is twice the number of elements in each of the k clusters.

The performance comparison of moUMDA and moUMDA employing K-means clustering is done with the well studied NSGA–II by Deb et al.[27]. NSGA–II uses non-dominated sorting and a crowding heuristic to break ties to make the best half of individuals become the parent population of the next generation. In a previous study [115], a modified version of NSGA–II has been discussed as a solution methodology for UAV path planning on a surveillance mission. The algorithm structure of NSGA–II adapted according to our problem definition is discussed in Algorithm 17.

An archive of the non-dominated solutions throughout all the generations is maintained externally for each algorithm to aid the convergence to the Pareto optimal front. This archive is updated with the best non-dominated solutions in each generation and any dominated solutions from previous generations are removed. Since, the feasible space is huge, the rationale behind maintaining this archive is to protect any good solutions that may get created at any stage of the optimisation process. When the algorithm is run for enough generations, with the help of dominance preservation in the archive, the archive will converge to the Pareto optimal front.

If the problem is non-noisy and deterministic in nature, the archive may be used to guide the search process [68]. If the archive becomes too large, then the size of the archive may also be limited to steer the search towards unexplored regions of the feasible by keeping the computation load reasonable. However, since we have a noisy problem in hand, maintaining a fixed size archive internally, would be more

Algorithm 17: NSGA–II for path planning of UAV in presence of noisy visibility

Initialise generation counter t=0;
Initialise random parent population $\mathcal{P}_Q^{parent}(t)$ of vertices of UAV path with population size λ ;
while *Termination condition not reached* **do**
  Create offspring population $\mathcal{P}_Q^{child}$ from $\mathcal{P}_Q^{parent}$;
  Combine parent and offspring population, $\mathcal{P}_Q^{parent} \cup \mathcal{P}_Q^{child} = \mathcal{P}_Q(t)$;
  Find shortest path *(Objective 1)* w.r.t. turning radius for each $p \in \mathcal{P}_\mathcal{Q}$;
  Calculate uncovered points w.r.t. S_i^{noisy} $\forall i$ for each $p \in \mathcal{P}_\mathcal{Q}$ along the shortest path with Bresenham Algorithm *(Objective 2)*;
  Sort population into fronts, *NoisyNDSorting($\mathcal{P}_\mathcal{Q}$)* $= (\mathcal{F}_1, \mathcal{F}_2, \dots)$;
  Initialise the next generation population $\mathcal{P}_Q(t+1) = \Phi$ and the front counter $i = 1$;
  while $|\mathcal{P}_Q(t+1)| + |\mathcal{F}_i| \leq \lambda$ **do**
    Include $\mathcal{F}_i$ in next population, $\mathcal{P}_Q(t+1) \cup \mathcal{F}_i = \mathcal{P}_Q(t+1)$;
    Check next front for inclusion $i = i + 1$;
  end
  Calculate $CrowdingDistance(\mathcal{F}_i)$;
  Sort $\mathcal{F}_i$ with respect to crowding distances;
  Include remaining individuals to $\mathcal{P}_Q(t+1)$, such that $\mathcal{P}_Q(t+1) = \mathcal{P}_Q(t+1) \cup \mathcal{F}_i[1 : (\lambda - |\mathcal{P}_Q(t+1)|)]$;
  Increment $t = t + 1$;
end

harmful than good, according to Chapter 5. In a noisy archive, the worse solutions may get added and good solutions may get discarded easily. In our scenario, since the visibility of the UAV with respect to the cloud coverage may change with time, this archive will be helpful to protect possible good solutions.

For estimation regarding successful assessment of the Pareto optimal set, the spread of a set of non-dominated solutions is measured with the help of the frequently used *hypervolume* performance indicator [124]. Here, since each of the objective functions are to be minimised, the hypervolume is measured by the area (or volume) of the region bounded below with a set of non-dominated candidate solutions and bounded above with a reference point r in the space of the objective functions, simultaneously. In our problem, the reference point r is given by the maximum value of each objective function can achieve in each corresponding dimension of the objective space, i.e., $r = (\max f_1, \max f_2, \ldots, \max f_k)$. When the hypervolume of a non-dominated solution set is high, it indicates that the non-dominated set is well-spread and has close proximity to the Pareto front.

6.3 Results and Discussions

A realistic scenario of UAV path planning problem in presence of weather factors is programmed in Matlab 2020b. The surveillance mission involves the coverage of a ground region of dimensions 150×120 units by a UAV. The visibility radius of the UAV when the weather factors are ideal is 10 units. The UAV follows a path through several vertices in an air grid at height h, i.e., the vertices are chosen from the $150 \times 120 = 18,000$ grid points. The UAV starts its mission from the southwest-most vertex on its path. The UAV can turn from one vertex to another with an angle greater than 120 degrees.

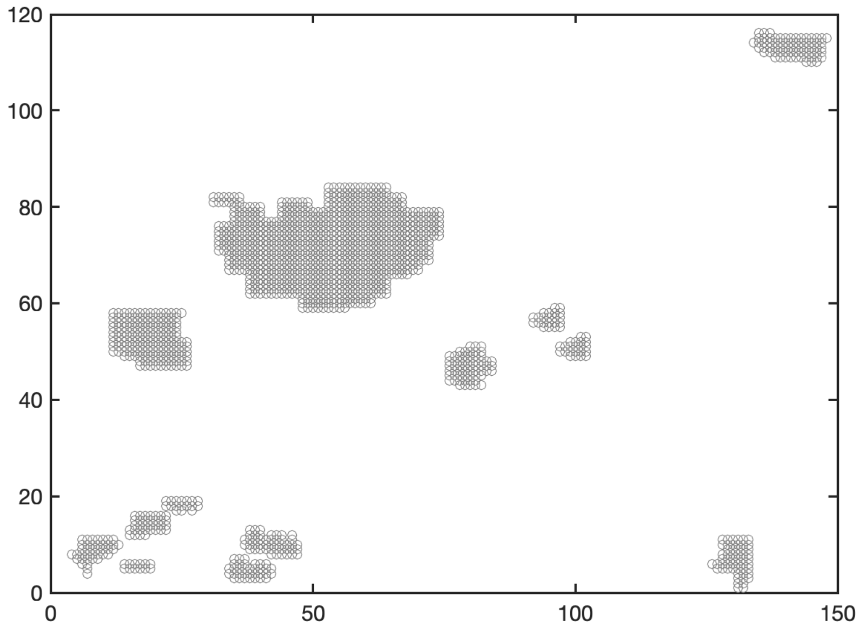

Figure 6.3: The considered cloud map C^{map}

In presence of weather factors, the visibility of the UAV is reduced. A cloud map C^{map} in the form of a Boolean matrix is considered which can be created from the weather reports of the area under consideration (see Figure 6.3). The area not visible due to cloud presence inside the visibility region of the UAV is represented as a noisy circle. The centre of the noisy circle is chosen at random within the visibility region and the radius is noisy in nature, i.e. $\mu + N(0, \sigma)$, where $\mu = 2$ units and $\sigma = 0.5$.

A population of $10 \times \log(150 \times 120)$ individuals for each algorithm is chosen, where every population individual is a grid with the randomly chosen vertices of the path of the UAV. For ensuring that the vertices of the UAV path are not too close to each other, the probability of having a one is 0.03 in each of the 18,000 grid points. To ensure parity in the running time of the algorithms and since the fitness function evaluation is the most computationally expensive stage, each of the algorithms are run until 3,000 fitness function evaluations are reached. Here, each fitness function evaluation involves finding the shortest feasible path through the vertices of the UAV path (as defined by an individual of the population) and calculating the amount of uncovered region if the UAV moves through that path.

Table 6.1: Hypervolume of final archive in 30 runs, when there is no cloud

Hypervolume of final archive	moUMDA	moUMDA +K-Means	NSGA–II
Best	8.9622×10^7	$\mathbf{8.9764 \times 10^7}$	5.8755×10^7
Average	8.9398×10^7	$\mathbf{8.9482 \times 10^7}$	5.6364×10^7
Worst	8.9185×10^7	$\mathbf{8.9217 \times 10^7}$	5.3436×10^7

Table 6.2: Hypervolume of final archive in 30 runs, when there are clouds

Hypervolume of final archive	moUMDA	moUMDA +K-Means	NSGA–II
Best	8.4974×10^7	$\mathbf{8.4988 \times 10^7}$	5.9703×10^7
Average	8.4720×10^7	$\mathbf{8.4846 \times 10^7}$	5.4336×10^7
Worst	8.4508×10^7	$\mathbf{8.4605 \times 10^7}$	5.0857×10^7

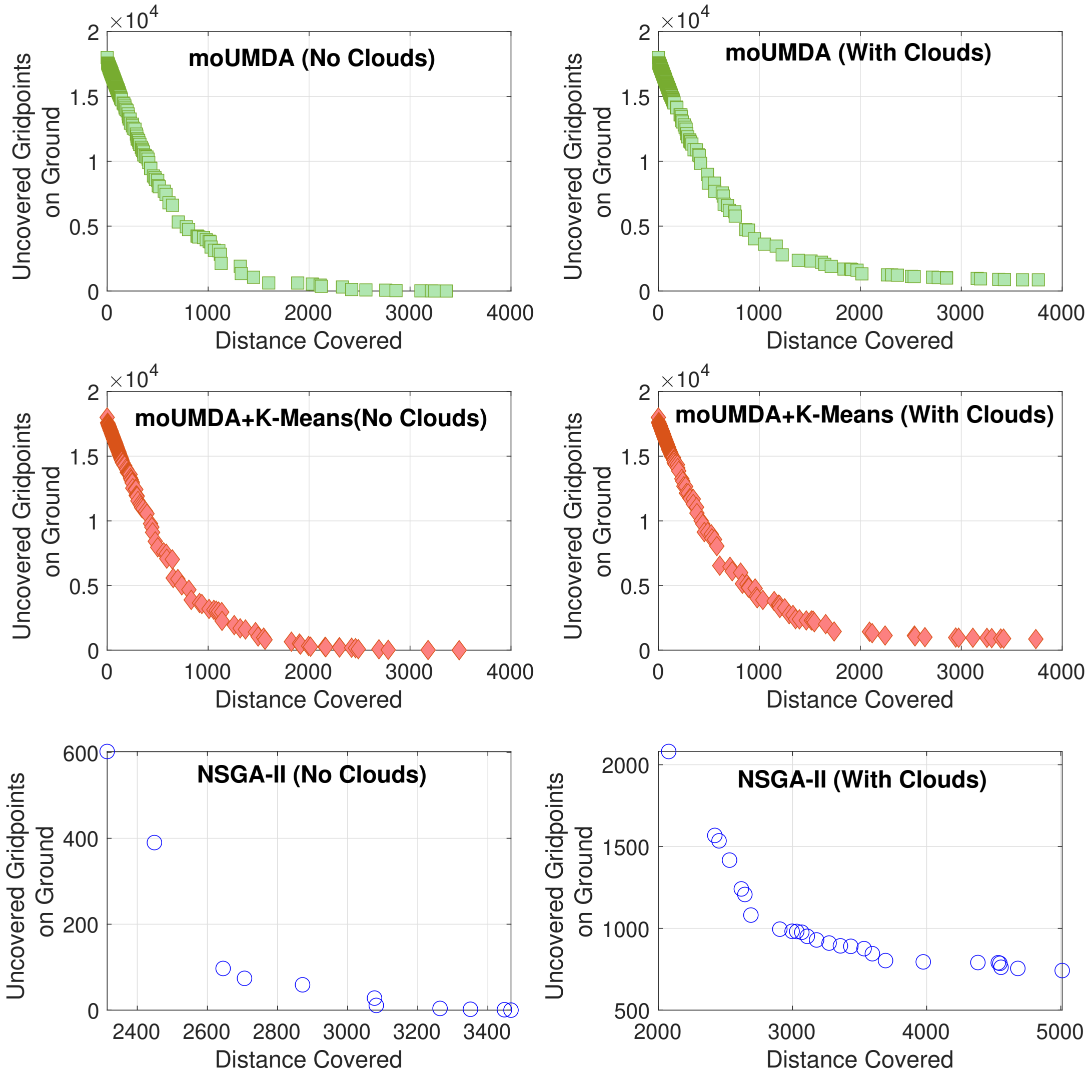

Figure 6.4: Pareto front corresponding from the final archive with the best hypervolume in 30 runs

Table 6.3: Average distance with respect to ground surveillance coverage, when there is no cloud

Coverage	moUMDA	moUMDA +K-Means	NSGA–II
100%	**3460**	**3417**	**3470**
95%-99%	**2059**	**2024**	2547
90%-94%	**1496**	**1488**	No data
80%-89%	1152	**1125**	No data
60%-79%	**762**	**761**	No data

Table 6.4: Average distance with respect to ground surveillance coverage, when there are clouds

Coverage	moUMDA	moUMDA +K-Means	NSGA–II
95%-99%	**3579**	**3614**	4047
90%-94%	**2206**	**2204**	2588
80%-89%	1409	**1363**	2187
60%-79%	**851**	**848**	No data

The hypervolume of the archive generated when the function evaluations budget is reached is calculated for each run of the algorithms. The archive that has the best hypervolume is plotted on the objective space for each of the algorithms in Figure 6.4. It is to be noted that the final archive is created on the basis of noisy fitness evaluations. Since noise is inherent in real-world problems, and the true (non-noisy) fitness cannot be accessed, the best value of algorithm performance indicator found until termination condition is reached is the compared for each of the algorithms.

In this problem, the best, average and the worst hypervolume of the archive within the 30 runs are reported in Tables 6.1 and 6.2. These results show that the variants of moUMDA (with the moUMDA version employing K-Means clustering being slightly better of the two), are a better choice to ensure a well-spread Pareto front. Even in the presence of noisy visibility due to clouds, the moUMDA variants are more effective. The best hypervolume in each category is marked with bold letters. The Mann-Whitney U-test has been used to ensure that the results are statistically significant at the 95% level. The results in Tables 6.1 and 6.2 are statistically significant.

For understanding the solution quality in terms of the second objective, i.e., the amount of ground coverage, Tables 6.3 and 6.4 are generated. The average distance required is reported on the basis of ground coverage percentage from the 30 runs. The better and statistically significant average distance (with the help of

Mann-Whitney U-test) from each category is marked with bold letters. In those cases when means of the ground coverage percentage within two algorithms are not significantly different, both the better performing algorithms are marked with bold letters. The variants of moUMDA provide a wide range of options to the user to help decide the required amount of coverage on the basis of distance covered by the UAV. It is clear that, in presence of clouds, the variants of moUMDA perform significantly better.

6.4 Summary

Realistic optimisation scenarios are often tampered with randomised uncertainties, however, real-world noisy combinatorial optimisation problems have not been studied much. A UAV path planning problem for ground surveillance is discussed, which becomes significantly harder to solve in presence of weather factors. The reduced visibility due to cloud presence is defined as a noisy circle within the visibility region of the UAV, where the noise is Gaussian with standard deviation σ. Two variants of multi-objective UMDA with and without K-Means clustering are proposed and compared with respect to the performance of the well-studied NSGA–II adapted to this UAV problem. The archive of non-dominated solutions from the moUMDA variants have a higher hypervolume than NSGA–II. In terms of ground coverage percentage in presence of random noise arising from weather factors, moUMDA variants are a better choice.

Chapter 7

Beyond EAs with Voting Algorithm

The promising performance of PCEA and UMDA employing two-parent and multi-parent recombination searching strategies respectively, imply that population based recombination techniques are important in handling noise. In this chapter, we study the use of voting mechanisms (employing bit-wise majority vote recombination) in a population of search strings. We introduce a new Voting algorithm that has superior performance in solving the OneMax problem in the presence of several noise models. The performance of the Voting algorithm is analysed on noisy linear and monotonic functions. Interestingly, implementing voting in population based algorithms like, UMDA, PCEA and cGA can result in significant performance speed-ups with a large population size.

7.1 Voting as a Recombination Strategy

In decision making problems, weighing in the decisions from multiple experts (classifiers) for a specific problem is a widely-used concept. Voting amongst the multiple decisions in order to achieve a unified decision is often a straightforward strategy, such as in the case of classifier ensembles. Voting methods help to combine several weak decisions to obtain a stronger decision [89].

In the context of evolutionary computation, a few studies have investigated voting strategies to perform multi-parent recombination, although recombination of multiple parents to generate an offspring is not known in natural evolution. An early survey of several multi-parent recombination strategies in evolutionary computation can be found in [36]. In this section, we review some of the previous literature which employ multi-parent recombination based on majority voting to generate the offspring.

An optimisation model known as the stochastic iterated genetic hillclimbing (SIGH) is studied by Ackley in 1987 [1]. This resembles an election system with active and apathetic voters electing a government. In each iteration, firstly, a search point of length n is generated during an election of the government by active voters in the system. In the next step of that iteration, if the active voters are unhappy with the chosen government i.e., the chosen search string, then they probably turn apathetic. In the last stage of the iteration, a reinforcement signal is generated based on the fitness evaluation of the current search string and it is compared with the history of function values. Based on the reinforcement signal, the active voters prepare their preferences to generate the search string (government) in the next iteration. The SIGH algorithm is reported as being better than GA with 1-point and uniform crossover and some hillclimbers in test function instances such as Trap and Porcupine with multiple local optima. The SIGH algorithm is shown as the slowest for graph partitioning problems.

A little later in 1989, a recombination operator known as the p-sexual voting recombination is studied for the quadratic assignment problem by Mühlenbein [80]. This operator generates a single offspring assignment by considering the majority vote of p parents. In the placements where majority is not attained, a mutation is considered. A parallel genetic algorithm employing this voting recombination within its neighbourhood is reported to perform better than the 2-opt algorithm for the assignment problem.

In a 1994 paper by Eiben et al. [37], several multi-parent recombination operators have been studied. A majority based recombination operator named as the occurrence-based scanning is defined and empirically analysed, that chooses the value of a bit position that occurs in the majority of the parents for the same bit position in the offspring. If none of the values occur more than the other, the value that is encountered first is added to the offspring bit position. They compare the performance of occurrence-based scanning with the other considered recombination for DeJong's test functions, TSP and graph colouring problems. Although multi-parent recombination operators performed well in the classical DeJong's test functions, in most of the cases, occurrence-based scanning did not perform well. The optimal performance was obtained for only 2 parents in case of occurrence-based scanning operator. This study was followed up in 1996 [38] by considering these multi-parent recombination operators in the GA framework for studying NK landscapes with low epistasis and it was shown that algorithms that employ recombination and mutation both are more efficient than mutation only algorithms.

The occurrence-based scanning operator is studied much later in recent literature by Friedrich et al. [46] and Whitley et al. [118] with 3 parents. This operator has been coined as the voting crossover and its efficacy for the JUMP function has been analysed. The results on the JUMP function are investigated further in the next subsection by proving simpler and better results.

7.1.1 Voting Crossover on the Jump Function

The JUMP function, introduced by Jansen & Wegener [65] is defined as follows,

$$\textsc{Jump}(x) = \begin{cases} m + |x|_1 & \text{if } |x|_1 \leq n - m \text{ or } |x|_1 = n \\ n - |x|_1 & \text{otherwise} \end{cases}$$

where m is the Hamming distance of the gap that must be jumped in order to find the optimum, and $|x|_1$ is the count of the number of ones in string x of length n. It is clear that the fitness value of a search point depends on the number bits that contain a one. The search strings which have the fitness $n-|x|_1$ are locally optimal and can be considered as a plateau within the search space. The global optimum is the all ones string. The gap between the plateau of the local optimum and the global optimum needs to be jumped.

Friedrich et al. [46] consider an initial phase of hillclimbing and a diversification mechanism with the *single receiver island model* proposed by [116], followed by a three parent voting crossover. This creates independent islands where hillclimbing phases with (1+1)–EA are run and the new search strings obtained from each island are combined with majority vote. They show that when diversity is maintained, the JUMP function can be solved in $O(n \log n)$ function evaluations for gap sizes, m as large as $O\left(n^{\frac{1}{2}-\epsilon}\right)$.

The algorithm proposed by Whitley et al. [118] for the JUMP problem proceeds in two phases and solves the JUMP problem, when $m = O(\log n)$, in linear time. In the first phase, a hill climber is used to get to a string containing $n-m$ ones and m zeros. They use the *next ascent bit climber* [25]. This works by choosing a random permutation of $\{1, \ldots, n\}$, and then mutating bits in that order, keeping changes that lead to improvements. After one round of this (which take n steps) we will have arrived at a string with $n-m$ ones. We mention in passing, that this approach would also solve ONEMAX in n steps. This process is repeated three times, starting each time from a fresh random string. The three resulting strings then perform the voting crossover — the offspring takes bit values given by a majority vote for each bit position. The whole process is repeated until the optimum is found.

Whitley et al. show that only a constant number of repeats is needed if the gap $m = O(\log n)$. However, we can improve on this (and simplify the proof) as follows. Since each of the three strings has begun from a fresh random initial string, the location of the m zeros in the resulting strings are independent, for each string. The vote will go wrong, in a single bit position, if either all three strings contain a zero, or only one of them contains a one at that position. The probability that this will happen is

$$\left(\frac{m}{n}\right)^3 + 3\left(\frac{m}{n}\right)^2\left(1 - \frac{m}{n}\right) \leq 3\left(\frac{m}{n}\right)^2$$

Thus, by the union bound, the probability that the vote goes wrong in at least one bit position, it is less than $3m^2/n$. So if $m < a\sqrt{n}$, where $a < 1/\sqrt{3}$ is a constant, then the vote fails with probability at most $3a^2$, meaning that we need to repeat the process an expected number of $1/(1 - 3a^2)$ times before the solution is found. Thus, the algorithm solves the Jump problem for gap sizes $m < \sqrt{n/3}$ in linear time.

7.2 The Voting Algorithm

Algorithm 18: The Voting Algorithm

```
Let p = (0, ..., 0);
repeat μ times
    Let x ∈ {0,1}^n be a random string;
    Let y ∈ {0,1}^n be a random string;
    if f(x) > f(y) then
        p = p + x;
    else
        p = p + y;
    end
end
for 1 ≤ i ≤ n do
    z_i = [p_i > μ/2];
end
Return z;
```

Now, we introduce a new heuristic algorithm based on applying a bit-wise majority vote to a population of search strings that has been produced by a selection mechanism. In this version, we use binary tournament selection. We generate two random strings, choose the best of the two, and add it to the population. When there are enough strings in the population, we take a bit-wise vote (see Algorithm 18).

7.3 Voting Algorithm on OneMax Problem

The performance of the voting algorithm is first analyzed for the OneMax problem (without noise). In order to do this, we need the following lemma.

Lemma 1. *Let $x, y \in \{0,1\}^n$ be random strings and let the tournament winner, z, be decided according to the* OneMax *function. For any $k \in \{1, \ldots n\}$,*

$$\Pr(z_k = 1) \geq \frac{1}{2} + \frac{1}{8\sqrt{n}}$$

Proof. The probability that the winner of the tournament has a one in position k is given by

$$\begin{aligned}\Pr\left(z_k = 1\right) &= \Pr\left(x_k = 1 \,|\, x \text{ wins}\right) \Pr\left(x \text{ wins}\right) \;+ \\ &\quad \Pr\left(y_k = 1 \,|\, y \text{ wins}\right) \Pr\left(y \text{ wins}\right)\end{aligned}$$

Then, by symmetry, we can obtain,

$$\Pr(z_k = 1) = \Pr(x_k = 1 \,|\, x \text{ wins}).$$

By Bayes' Theorem

$$\Pr(x_k = 1 \,|\, x \text{ wins}) = \frac{\Pr(x_k = 1)}{\Pr(x \text{ wins})} \Pr(x \text{ wins} \,|\, x_k = 1)$$

Since $\Pr(x_k = 1) = \Pr(x \text{ wins}) = \frac{1}{2}$, we can write the following,

$$\Pr(x_k = 1 \,|\, x \text{ wins}) = \Pr(x \text{ wins} \,|\, x_k = 1)$$

Then, by the law of total probability,

$$\begin{aligned}
& \Pr(x_k = 1 \,|\, x \text{ wins}) \\
= \; & \Pr(x \text{ wins} \,|\, x_k = 1, y_k = 1) \Pr(y_k = 1) + \Pr(x \text{ wins} \,|\, x_k = 1, y_k = 0)) \Pr(y_k = 0) \\
= \; & \frac{1}{2} \Pr(x \text{ wins} \,|\, x_k = 1, y_k = 1) + \frac{1}{2} \Pr(x \text{ wins} \,|\, x_k = 1, y_k = 0) \\
\geq \; & \frac{1}{4} + \frac{1}{2} \Pr\left(\sum_{i \neq k} x_i + 1 > \sum_{i \neq k} y_i\right) \\
= \; & \frac{1}{4} + \frac{1}{2} \Pr\left(\sum_{i \neq k} y_i - x_i < 1\right) \\
= \; & \frac{1}{4} + \frac{1}{2} \left(\Pr\left(\sum_{i \neq k} y_i - x_i < 0\right) + \Pr\left(\sum_{i \neq k} y_i - x_i = 0\right)\right)
\end{aligned}$$

Since,

$$\Pr\left(\sum_{i \neq k} y_i - x_i < 0\right) + \Pr\left(\sum_{i \neq k} y_i - x_i = 0\right) + \Pr\left(\sum_{i \neq k} y_i - x_i > 0\right) = 1$$

By symmetry, we have, $\Pr\left(\sum_{i \neq k} y_i - x_i < 0\right) = \Pr\left(\sum_{i \neq k} y_i - x_i > 0\right)$, so,

$$\Pr\left(\sum_{i \neq k} y_i - x_i < 0\right) = \frac{1}{2}\left(1 - \Pr\left(\sum_{i \neq k} y_i - x_i = 0\right)\right)$$

so that

$$\begin{aligned}
& \Pr(x_k = 1 \,|\, x \text{ wins}) \\
= \; & \frac{1}{2} + \frac{1}{4}\left(\Pr\left(\sum_{i \neq k} y_i - x_i = 0\right)\right)
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{2} + \frac{1}{4}\left(\sum_{j=0}^{n-1} \Pr\left(\sum_{i\neq k} y_i = j\right) \Pr\left(\sum_{i \neq k} x_i = j\right)\right) \\
&= \frac{1}{2} + \frac{1}{2^{2n}} \sum_{j=0}^{n-1} \binom{n-1}{j}^2 \\
&= \frac{1}{2} + \frac{1}{2^{2n}} \binom{2n-2}{n-1} \\
&\geq \frac{1}{2} + \frac{1}{8\sqrt{n}}
\end{aligned}
$$

where the final inequality derives from [110]. □

We can now show that the voting algorithm solves OneMax (with high probability) if $\mu = \Omega(n \log n)$.

Theorem 1. *If $\mu = 32(c+1)n \log n$, then the Voting algorithm correctly solves* OneMax *with probability greater than $1 - 1/n^c$.*

Proof. For any one bit position, k, the probability that the vote is incorrect is

$$
\Pr(p_k \leq \mu/2) \leq \exp(-2\mu/64n) = \frac{1}{n^{c+1}}
$$

by Hoeffding's inequality. So by the union bound, the probability that at least one bit gets the incorrect vote is at most $1/n^c$. □

7.3.1 Strings Not Chosen Uniformly at Random

The analysis and experiments with EAs usually involve starting with search string(s) chosen uniformly at random, where the probability of having one in each bit is set to $\frac{1}{2}$. If generalised initialisation of strings are considered, such that each bit of the random strings would have a one with probability r and a zero otherwise, then it enables the analysis of the different scenarios arising from the arbitrariness of r.

The different situations being as follows – when the initial strings are better than random, i.e., $r > \frac{1}{2}$, or, when the strings are chosen uniformly at random, i.e.,

$r = \frac{1}{2}$, as well as, when the strings are worse than random, i.e., $r < \frac{1}{2}$. However, considering $r \geq \frac{1}{2}$ is of concern, in general.

The consideration of the probability r instead of choosing the strings uniformly at random, is an initial step towards understanding the impact of biasing the initial population, theoretically, which is needed in certain application domains (as considered in the UAV problem in Chapter 6).

When the voting mechanism is hybridised with an initial phase of local search, such as $(1 + 1)$–EA, or in other population-based algorithms (refer to empirical section of this chapter) to generate population, the search strings in the population tend to be closer to optimum than random search strings chosen with binary tournament selection. This would mean that a smaller population would be needed to reach the optimum.

Voting mechanisms are often employed in machine learning frameworks such as obtaining decisions from classifier ensembles. In the case where the bits of a search string are considered as data attributes of learning problems, theoretical analysis would require an assumption that each attribute has a specific probability of having a one (success). Although, in realistic scenarios, the probability of success in some of the attributes will be correlated, a simplified assumption may be to consider that the attributes are independent.

Lemma 2. *Let $x, y \in \{0, 1\}^n$ be two strings chosen at random, but not uniformly. The probability probability of having a one in each bit of x and y is r.*

The winner of the binary tournament selection, z decided on the basis of OneMax *function, will have a one in bit position k with probability*

$$\Pr(z_k = 1) \geq r + r(1 - r)\frac{1}{2\sqrt{n}}$$

Proof. The probability that the winner of the tournament has a one in position k

is given by

$$\Pr(z_k = 1) = \Pr(x_k = 1 \,|\, x \text{ wins}) \Pr(x \text{ wins}) + \Pr(y_k = 1 \,|\, y \text{ wins}) \Pr(y \text{ wins})$$

Then, by symmetry, we can obtain,

$$\Pr(z_k = 1) = \Pr(x_k = 1 \,|\, x \text{ wins})$$

By Bayes' Theorem

$$\begin{aligned} \Pr(x_k = 1 \,|\, x \text{ wins}) &= \frac{\Pr(x_k = 1)}{\Pr(x \text{ wins})} \Pr(x \text{ wins} \,|\, x_k = 1) \\ &= 2r \times \Pr(x \text{ wins} \,|\, x_k = 1) \end{aligned}$$

Using the law of total probability, we can say that,

$$\begin{aligned} & \Pr(x \text{ wins} \,|\, x_k = 1) \\ = \; & \Pr(x \text{ wins} \,|\, x_k = 1, y_k = 1) \Pr(y_k = 1) + \Pr(x \text{ wins} \,|\, x_k = 1, y_k = 0) \Pr(y_k = 0) \\ \geq \; & \frac{1}{2} r + \Pr\left(\sum_{i \neq k} x_i + 1 > \sum_{i \neq k} y_i \right) (1 - r) \\ \geq \; & \frac{1}{2} r + \Pr\left(\sum_{i \neq k} (y_i - x_i) < 1 \right) (1 - r) \\ = \; & \frac{1}{2} r + (1 - r) \left[\Pr\left(\sum_{i \neq k} (y_i - x_i) < 0 \right) + \Pr\left(\sum_{i \neq k} (y_i - x_i) = 0 \right) \right] \end{aligned}$$

By symmetry, we have

$$\Pr\left(\sum_{i \neq k} (y_i - x_i) < 0 \right) = \frac{1}{2} \left(1 - \Pr\left(\sum_{i \neq k} (y_i - x_i) = 0 \right) \right)$$

Now, we can obtain,

$$
\begin{aligned}
& \Pr\left(\sum_{i\neq k}(y_i - x_i) < 1\right) \\
= \quad & \frac{1}{2} + \frac{1}{2}\Pr\left(\sum_{i\neq k}(y_i - x_i) = 0\right) \\
= \quad & \frac{1}{2} + \frac{1}{2}\left(\sum_{j=0}^{n-1}\Pr\left(\sum_{i\neq k} y_i = j\right)\Pr\left(\sum_{i\neq k} x_i = j\right)\right) \\
= \quad & \frac{1}{2} + \frac{1}{2}\sum_{j=0}^{n-1}\binom{n-1}{j}^2 r^{2j}(1-r)^{2n-2-2j}
\end{aligned}
$$

Since the above summation takes the form of the well-known Bernstein polynomial, using a conjecture on the squared Bernstein polynomials (found in Conjecture 1 in [50] and Lemma 3.5 in [56]), we can obtain the following inequality,

$$\geq \frac{1}{2} + \frac{1}{2^{2n-1}}\binom{2n-2}{n-1}$$

The above may be bounded using the work by Stanica [110], to obtain as follows,

$$\geq \frac{1}{2} + \frac{1}{4\sqrt{n}}$$

Then the required probability of having a one in the bit position k of the tournament winner z is derived as follows,

$$
\begin{aligned}
\Pr(z_k = 1) \quad & \geq \quad r^2 + 2r(1-r)\left(\frac{1}{2} + \frac{1}{4\sqrt{n}}\right) \\
& = \quad r + r(1-r)\frac{1}{2\sqrt{n}}
\end{aligned}
$$

□

Theorem 2. *If $\mu = \frac{(c+1)}{2t^2}\log n$, then the Voting algorithm with $r = \frac{1}{2} + t$, where $t > 0$ is a constant, correctly solves* OneMax *with probability greater than $1 - 1/n^c$.*

More generally, $O(\log n)$ function evaluations are required by the Voting Algorithm to solve the ONEMAX*, when $r = \frac{1}{2} + t$.*

Proof. When $r = \frac{1}{2} + t$,

$$\begin{aligned}\Pr(z_k = 1) &\geq r + r(1-r)\frac{1}{2\sqrt{n}} \\ &\geq r, \quad \text{since both terms are positive quantities}\end{aligned}$$

Also, when $n \to \infty$, the second term vanishes.

Now, by the Hoeffding's inequality, for any bit position k we get the following

$$\begin{aligned}\Pr\left(p_k \leq \frac{\mu}{2}\right) &= \Pr\left(p_k \leq \left(r\mu - r\mu + \frac{\mu}{2}\right)\right) \\ &\leq \exp\left(-2\left(r - \frac{1}{2}\right)^2 \mu\right) \\ &= \exp(-2\mu t^2) = \frac{1}{n^{c+1}}\end{aligned}$$

Now, by the union bound it follows that the probability of having at least one bit that gets the incorrect vote is $1/n^c$.

□

We know that, the most efficient algorithm [7] would require a running time of $\Theta(n/\log n)$ to solve the ONEMAX. However, when fitness comparisons are used, the best-case scenario would require at least n comparisons. The above theorem shows that the Voting algorithm requires only $\Omega(\log n)$ function evaluations to reach the optimum in solving ONEMAX when the initial strings are better (closer to optimum) than random strings.

Theorem 3. *If $\mu = 32(c+1)n\log n$, then the Voting algorithm with $r = \frac{1}{2} + \epsilon_n$, where $\epsilon_n \to 0$ as $n \to \infty$, correctly solves* ONEMAX *with probability greater than $1 - 1/n^c$. More generally, $O(n \log n)$ function evaluations are required by the Voting Algorithm to solve the* ONEMAX*, when $r = \frac{1}{2} + \epsilon_n$.*

Proof. The probability that the tournament winner has a one in bit position k,

$$\begin{aligned}
\Pr(z_k = 1) &\geq \left(\frac{1}{2} + \epsilon_n\right) + \left(\frac{1}{2} + \epsilon_n\right)\left(\frac{1}{2} - \epsilon_n\right)\frac{1}{2\sqrt{n}} \\
&= \frac{1}{2} + \epsilon_n + \frac{1}{8\sqrt{n}} - \frac{{\epsilon_n}^2}{2\sqrt{n}} \\
&\geq \frac{1}{2} + \frac{1}{8\sqrt{n}}, \quad \text{for sufficiently large } n.
\end{aligned}$$

□

The above theorem discusses the scenario when n becomes sufficiently large and the strings are chosen at random with a probability infinitesimally close to $\frac{1}{2}$.

This interesting idea of considering a non-uniform random string initialisation may be considered for the following analyses. However, the illustration of this has been avoided in the following analyses, as the main focus of this chapter is to understand the performance of Voting algorithm in presence of several noise variants and in solving linear functions.

7.4 Voting Algorithm on Jump Function

When initially the strings are chosen uniformly at random, it is clear that the Voting algorithm only samples strings which have a number of ones close to $n/2$. Indeed by Hoeffding's inequality it is exponentially unlikely to sample strings with greater than $n/2 + \alpha n$ ones, for any constant $1/2 < \alpha < 1$. We get the following lemma,

Lemma 3. *If initially, the strings are chosen uniformly at random, and* $\mu = 32(c+1)n \log n$*, then the Voting algorithm correctly solves* JUMP *with probability greater than* $1 - 1/n^c$*, for any gap size* $m < (1-\alpha)n$ *where* α *is a constant in the range* $1/2 < \alpha < 1$.

7.5 Voting Algorithm on Noisy OneMax

The next subsections discuss the runtime bounds for the OneMax in presence of different noise variants.

7.5.1 Posterior Additive Noise

For the OneMax problem with posterior noise, the fitness, at each evaluation, receives an addition of a random value drawn from some probability distribution η with variance σ^2.

$$f^{noisy}(x) = f(x) + \eta(\sigma^2)$$

We will show the voting algorithm has superior runtime than other existing algorithms (refer to Table 2.3) on noisy OneMax for arbitrary posterior noise distributions with the only consideration that the noise is unimodal in nature. We will assume that the noise distribution has finite mean and variance σ^2 .

In the proof above for OneMax with no noise, we used a bound on the central binomial coefficient. We now need a similar bound for binomial coefficients that are close to the centre.

Lemma 4. *For any integers $m > 0$ and $0 \leq k \leq \sqrt{m}$ we have*

$$\binom{2m}{m+k} \geq \left(\frac{2\sqrt{\pi}}{e^4}\right) \frac{2^{2m}}{\sqrt{m}}$$

Proof. We use the following inequalities associated with Stirling's approximation:

$$\sqrt{2\pi} n^{n+1/2} e^{-n} \leq n! \leq e n^{n+1/2} e^{-n}$$

to give

$$\binom{2m}{m+k} = \frac{(2m)!}{(m+k)!(m-k)!}$$

$$
\begin{aligned}
&\geq \left(\frac{\sqrt{2\pi}}{e^2}\right)\frac{2^{2m+1/2}m^{2m+1/2}}{(m+k)^{m+k+1/2}(m-k)^{m-k+1/2}}\\
&= \left(\frac{2\sqrt{\pi}}{e^2}\right)\left(\frac{2^{2m}}{\sqrt{m}}\right)\frac{1}{(1+k/m)^{m+k+1/2}(1-k/m)^{m-k+1/2}}\\
&= \left(\frac{2\sqrt{\pi}}{e^2}\right)\left(\frac{2^{2m}}{\sqrt{m}}\right)\frac{1}{(1-k^2/m^2)^{m-k+1/2}(1+k/m)^{2k}}\\
&\geq \left(\frac{2\sqrt{\pi}}{e^2}\right)\left(\frac{2^{2m}}{\sqrt{m}}\right)\frac{1}{(1+k/m)^{2k}}\\
&\geq \left(\frac{2\sqrt{\pi}}{e^4}\right)\frac{2^{2m}}{\sqrt{m}}
\end{aligned}
$$

□

We will also need the following.

Lemma 5. *Given two random binary strings, a, b, of length m, and any integer $0 \leq s \leq m$, we have*

$$\Pr(|a|_1 - |b|_1 = s) = \frac{1}{2^{2m}}\binom{2m}{m+s}$$

Proof. To achieve a difference of s requires picking $i \geq s$ ones in string a and then $i - s$ ones in string b. The probability this happens is

$$\frac{1}{2^{2m}}\sum_{i=s}^{m}\binom{m}{i}\binom{m}{i-s}$$

Now the number of ways of choosing $m+s$ items from $2m$ items can be described by the number of ways of choosing at least s items from the first m, and then the remainder from the other m items. That is

$$\binom{2m}{m+s} = \sum_{i=s}^{m}\binom{m}{i}\binom{m}{m+s-i} = \sum_{i=s}^{m}\binom{m}{i}\binom{m}{i-s}$$

by the symmetry of binomial coefficients. The result follows. □

Theorem 4. *The Voting algorithm correctly solves noisy* OneMax *with high prob-*

ability, when the noise distribution has finite mean and variance $\sigma^2 \leq 3n/8$, in $O(n \log n)$ function evaluations.

If, in addition, the noise distribution is unimodal, then in the case $\sigma^2 \geq 3n/8$, the algorithm requires $O(\sigma^2 \log n)$ function evaluations.

Proof. As with the analysis of OneMax without noise, the probability that the winner of a tournament between two random strings x and y has a one in position k is equal to

$$\begin{aligned}
& \Pr(x_k = 1 \,|\, x \text{ wins}) \\
= \; & \frac{1}{2} \Pr(x \text{ wins} \,|\, x_k = 1, y_k = 1) + \frac{1}{2} \Pr(x \text{ wins} \,|\, x_k = 1, y_k = 0) \\
\geq \; & \frac{1}{4} + \frac{1}{2} \Pr\left(\sum_{i \neq k} x_i + 1 + U > \sum_{i \neq k} y_i + V \right) \\
= \; & \frac{1}{4} + \frac{1}{2} \Pr\left(\sum_{i \neq k} y_i - x_i < 1 + U - V \right)
\end{aligned}$$

where U and V are independent random samples from the noise distribution.

We note that if U and V are two independent random values drawn from the noise distribution, then $U - V$ comes from a symmetric distribution with zero mean, and variance $2\sigma^2$. Now,

$$\begin{aligned}
& \Pr\left(\sum_{i \neq k} y_i - x_i < 1 + U - V \right) \\
& = \sum_{s=-(n-1)}^{n-1} \Pr\left(\sum_{i \neq k} y_i - x_i = s \right) \Pr(U - V > s - 1) \\
& = \Pr\left(\sum_{i \neq k} y_i - x_i = 0 \right) \Pr(U - V > -1) \\
& + \sum_{s=1}^{n-1} \Pr\left(\sum_{i \neq k} y_i - x_i = s \right) \left(\Pr(s - 1 < U - V) + \Pr(U - V < s + 1) \right)
\end{aligned}$$

(where we have used the fact that $U - V$ is symmetric)

$$= \Pr\left(\sum_{i \neq k} y_i - x_i = 0\right) \Pr(U - V > -1) +$$

$$\sum_{s=1}^{n-1} \Pr\left(\sum_{i \neq k} y_i - x_i = s\right) (1 + \Pr(s - 1 < U - V < s + 1))$$

$$= \Pr\left(\sum_{i \neq k} y_i - x_i = 0\right) \Pr(U - V > -1) + \frac{1}{2} - \frac{1}{2} \Pr\left(\sum_{i \neq k} y_i - x_i = 0\right)$$

$$+ \sum_{s=1}^{n-1} \Pr\left(\sum_{i \neq k} y_i - x_i = s\right) \Pr(s - 1 < U - V < s + 1)$$

$$= \frac{1}{2} + \Pr\left(\sum_{i \neq k} y_i - x_i = 0\right) \Pr(0 < U - V < 1)$$

$$+ \sum_{s=1}^{n-1} \Pr\left(\sum_{i \neq k} y_i - x_i = s\right) \Pr(s - 1 < U - V < s + 1)$$

$$\geq \frac{1}{2} + \Pr\left(\sum_{i \neq k} y_i - x_i = 0\right) \Pr(0 < U - V < 1)$$

$$+ \sum_{s=1}^{\lfloor\sqrt{n}\rfloor} \Pr\left(\sum_{i \neq k} y_i - x_i = s\right) \Pr(s - 1 < U - V < s + 1)$$

$$\geq \frac{1}{2} + \left(\frac{2\sqrt{\pi}}{e^4\sqrt{n}}\right) \Pr(|U - V| \leq \lfloor\sqrt{n}\rfloor - 1)$$

where we have used the preceding lemmas. Thus, the probability of a one appearing in bit position k in the tournament winner is at least

$$\frac{1}{2} + \left(\frac{\sqrt{\pi}}{e^4\sqrt{n}}\right) \Pr(|U - V| \leq \lfloor\sqrt{n}\rfloor - 1)$$

In the case where $\sigma^2 \leq 3n/8$, we can use Chebyshev's inequality:

$$\Pr(|U - V| \leq \lfloor\sqrt{n}\rfloor - 1) \geq 1 - \frac{2\sigma^2}{(\lfloor\sqrt{n}\rfloor - 1)^2}$$

to show that the probability of a one appearing in position k is at least

$$\frac{1}{2} + \left(\frac{\sqrt{\pi}}{100e^4\sqrt{n}}\right)$$

for sufficiently large n. Following the same argument as in the non-noisy case allows us to conclude that the runtime is $O(n \log n)$ with high probability.

When σ^2 is larger, we additionally assume that the noise distribution is unimodal, from which it follows that the distribution of $U - V$ is also unimodal [114]. We may then use the Camp-Meidell inequality:

$$\Pr(|U - V| \leq \lfloor\sqrt{n}\rfloor - 1) \geq \frac{\lfloor\sqrt{n}\rfloor - 1}{\sqrt{6}\sigma}$$

which shows the probability of a one in position k to be at least:

$$\frac{1}{2} + \frac{\sqrt{\pi}}{4\sqrt{3}e^4\sigma}$$

for sufficiently large n. The inequality is valid for $\sigma^2 > 3n/8$. The remainder of the proof follows as before. □

7.5.2 Prior Bit-flipping Noise

The prior noise flips a single bit or multiple bits in the search point before the fitness evaluation is performed. Here, we have considered the generalised multiple bit-flipping noise, that is defined as follows,

$$f^{noisy}(x) = \begin{cases} f(x) & \text{with probability } (1-p) \\ f(x') & \text{with probability } p \end{cases}$$

where, x' is generated by independently flipping each bit of x with probability q.

When the OneMax problem is affected by the prior bit-flipping noise, a bit gets flipped with probability pq. In this work, we present a better bound than the existing results (refer to Table 2.1) for this problem.

Theorem 5. *The Voting algorithm correctly solves noisy* OneMax *in the presence* *of the generalised multiple bit-flipping noise, with high probability, when $pq < \frac{2}{5}$, in $O(n \log n)$ function evaluations.*

Proof. As with the analysis of OneMax without noise, the probability that the winner of a tournament between two strings x and y, chosen uniformly at random, has one in position k is equal to

$$\Pr(z_k = 1) = \Pr(x_k = 1 \mid x \text{ wins}) = \Pr(x \text{ wins} \mid x_k = 1)$$

The cases that the bit k is flipped and is not flipped are denoted by $\not\mathcal{F}(\text{bit } k)$ and $\mathcal{F}(\text{bit } k)$ respectively. Then,

$$\Pr(\not\mathcal{F}(\text{bit } k)) = pq$$
$$\Pr(\mathcal{F}(\text{bit } k)) = (1 - pq)$$

There can be two possibilities: $y_k = 1$ or $y_k = 0$ and for each of these, four different situations may arise dependent on flipping of x_k and y_k, illustrated as follows,

Case 1: $\Pr(y_k = 1, \not\mathcal{F}(x_k), \not\mathcal{F}(y_k)) = \frac{1}{2}(pq)^2$

Case 2: $\Pr(y_k = 1, \not\mathcal{F}(x_k), \mathcal{F}(y_k)) = \frac{1}{2}pq(1 - pq)$

Case 3: $\Pr(y_k = 1, \mathcal{F}(x_k), \not\mathcal{F}(y_k)) = \frac{1}{2}(1 - pq)pq$

Case 4: $\Pr(y_k = 1, \mathcal{F}(x_k), \mathcal{F}(y_k)) = \frac{1}{2}(1 - pq)^2$

Case 5: $\Pr(y_k = 0, \not\mathcal{F}(x_k), \not\mathcal{F}(y_k)) = \frac{1}{2}(pq)^2$

Case 6: $\Pr(y_k = 0, \not\mathcal{F}(x_k), \mathcal{F}(y_k)) = \frac{1}{2}pq(1 - pq)$

Case 7: $\Pr(y_k = 0, \mathcal{F}(x_k), \not\mathcal{F}(y_k)) = \frac{1}{2}(1 - pq)pq$

Case 8: $\Pr(y_k = 0, \mathcal{F}(x_k), \mathcal{F}(y_k)) = \frac{1}{2}(1 - pq)^2$

By conditioning $\Pr(x \text{ wins} \mid x_k = 1)$ with respect to the above cases the required probability may be derived as follows,

$$\Pr(x \text{ wins} \mid x_k = 1)$$

$$
\begin{aligned}
= &\ \frac{1}{2}p^2q^2\,\Pr(x \text{ wins} \mid x_k = 1, \textit{Case 1}) \\
+ &\ \frac{1}{2}pq(1-pq)\,\Pr(x \text{ wins} \mid x_k = 1, \textit{Case 2}) \\
+ &\ \frac{1}{2}pq(1-pq)\,\Pr(x \text{ wins} \mid x_k = 1, \textit{Case 3}) \\
+ &\ \frac{1}{2}(1-pq)^2\,\Pr(x \text{ wins} \mid x_k = 1, \textit{Case 4}) \\
+ &\ \frac{1}{2}p^2q^2\,\Pr(x \text{ wins} \mid x_k = 1, \textit{Case 5}) \\
+ &\ \frac{1}{2}pq(1-pq)\,\Pr(x \text{ wins} \mid x_k = 1, \textit{Case 6}) \\
+ &\ \frac{1}{2}pq(1-pq)\,\Pr(x \text{ wins} \mid x_k = 1, \textit{Case 7}) \\
+ &\ \frac{1}{2}(1-pq)^2\,\Pr(x \text{ wins} \mid x_k = 1, \textit{Case 8})
\end{aligned}
$$

This may be simplified as follows,

$$
\begin{aligned}
= &\ \frac{1}{2}p^2q^2 \times \Pr(x \text{ wins} \mid x_k = 0, y_k = 0) \\
+ &\ \frac{1}{2}pq(1-pq) \times \Pr(x \text{ wins} \mid x_k = 0, y_k = 1) \\
+ &\ \frac{1}{2}pq(1-pq) \times \Pr(x \text{ wins} \mid x_k = 1, y_k = 0,) \\
+ &\ \frac{1}{2}(1-pq)^2 \times \Pr(x \text{ wins} \mid x_k = 1, y_k = 1) \\
+ &\ \frac{1}{2}p^2q^2 \times \Pr(x \text{ wins} \mid x_k = 0, y_k = 1) \\
+ &\ \frac{1}{2}pq(1-pq) \times \Pr(x \text{ wins} \mid x_k = 0, y_k = 0) \\
+ &\ \frac{1}{2}pq(1-pq) \times \Pr(x \text{ wins} \mid x_k = 1, y_k = 1) \\
+ &\ \frac{1}{2}(1-pq)^2 \times \Pr(x \text{ wins} \mid x_k = 1, y_k = 0)
\end{aligned}
$$

We know, that when $x_k = y_k$, there is an equal probability in obtaining the winning total by the either strings. The probability that $\Pr(x \text{ wins} \mid x_k = 0, y_k = 1)$ has been calculated during the analysis of OneMax without noise. With the help of the following calculation, we can obtain the required probability.

$$
\begin{aligned}
& \Pr(x \text{ wins} \mid x_k = 0, y_k = 1) \\
= \;& \Pr\left(\sum_{i \neq k} (x_i - y_i) > 1\right) \\
= \;& 1 - \Pr\left(\sum_{i \neq k} (x_i - y_i) <= 1\right) \\
= \;& 1 - \Pr\left(\sum_{i \neq k} (x_i - y_i) \leq 0\right) - \Pr\left(\sum_{i \neq k} (x_i - y_i) = 1\right) \\
\geq \;& 1 - \Pr\left(\sum_{i \neq k} (x_i - y_i) \leq 0\right) - \Pr\left(\sum_{i \neq k} (x_i - y_i) = 0\right) \\
= \;& \frac{1}{2} - \frac{3}{2} \Pr\left(\sum_{i \neq k} (x_i - y_i) = 0\right)
\end{aligned}
$$

Then the probability that the winner has a one in position k may be obtained as,

$$\Pr(z_k = 1) \geq \frac{1}{2} + \frac{1}{4\sqrt{n}} - \frac{5}{8\sqrt{n}} pq$$

Proceeding as before by considering the union bound and Hoeffding's inequality, we can say that the voting algorithm solves OneMax with high probability in presence of bitwise prior noise when $\left(\frac{1}{4} - \frac{5}{8}pq\right) > 0$, which means that $pq < \frac{2}{5}$, if $\mu = O(n \log n)$. □

Theorem 6. *If* $\mu = \frac{8(c+1)n \log n}{\left(1 - \frac{5}{2}pq\right)^2}$, *then the Voting algorithm correctly solves* OneMax *in the presence of prior bit-flipping noise with probability greater than* $1 - 1/n^c$, *given,* $pq < \frac{2}{5}$.

Proof. For any bit position k, the probability that the vote is incorrect for $r = \frac{1}{2}$ is given by,

$$\Pr(p_k \leq \mu/2) \leq \exp\left(\frac{-\mu\left(1 - \frac{5}{2}pq\right)^2}{8n}\right) = \frac{1}{n^{c+1}}$$

by Hoeffding's inequality. So by the union bound, the probability that at least one bit gets the incorrect vote is at most $1/n^c$. □

7.5.3 Partial Evaluation of Fitness Function

In noisy data mining and learning problems, incomplete or unavailable data attributes are often encountered. Here, we have considered a similar noise model where, the fitness evaluation is performed only on a random subset of bits (attributes) of the search point which leads to a partial evaluation of the fitness function. This noise model has been studied for a non-elitist binary selection algorithm and $(1+1)$–EA [23].

When solving the $\textsc{OneMax}(x)$ function, where the noisy evaluation takes into consideration the ones with a probability d, and otherwise with a probability $(1-d)$. The noisy fitness function can be written as follows,

$$f^{noisy}(x) = \sum_{i=1}^{n} a_i x_i = \begin{cases} f(x) & \text{with probability } d^{f(x)}(1-d)^0 \\ f(x)-1 & \text{with probability } d^{(f(x)-1)}(1-d)^1 \\ f(x)-2 & \text{with probability } d^{(f(x)-2)}(1-d)^2 \\ \dots & \\ 0 & \text{with probability } d^0(1-d)^{f(x)} \end{cases}$$

where, a_i is the indicator function that depends on whether x_i is a one and if it is considered in the fitness evaluation with a probability d.

Here, we show that (in section 4.3) Voting algorithm solves the $\textsc{OneMax}$ problem with partial evaluation of fitness evaluation in $O(n \log n)$ function evaluations, however, according to [23], a non-elitist binary tournament EA would require a much larger bound.

Theorem 7. *Let $x, y \in \{0,1\}^n$ be two strings chosen uniformly at random. The* *winner of the binary tournament selection on the* OneMax *function, with partial evaluation of fitness function with probability d, then the probability that z will have a one in position k is given by*

$$\Pr(z_k = 1) \geq \frac{1}{2} + \frac{d}{8\sqrt{n}}$$

If $\mu = \frac{32}{d^2}(c+1)n\log n$, then the Voting algorithm correctly solves OneMax *with probability greater than $1 - 1/n^c$.*

Proof. Following as before,

$$\begin{aligned}
\Pr(z_k = 1) &= \Pr(x_k = 1 \,|\, x \text{ wins}) \\
&= \Pr(x \text{ wins} \,|\, x_k = 1) \\
&= \Pr(x \text{ wins} \,|\, x_k = 1, y_k = 1) \Pr(y_k = 1) \\
&+ \Pr(x \text{ wins} \,|\, x_k = 1, y_k = 0) \Pr(y_k = 0) \\
&= \frac{1}{4} + \frac{1}{2} \Pr(x \text{ wins} \,|\, x_k = 1, y_k = 0)
\end{aligned}$$

Here, for each i, let a_i and b_i be random numbers such that they are equal to one with probability d when, bit position i is being considered in the fitness evaluation and zero with probability $(1-d)$, otherwise.

Then we can say,

$$\Pr(x \text{ wins} \,|\, x_k = 1, y_k = 0) \geq \Pr\left(\sum_{i \neq k} a_i x_i + a_k > \sum_{i \neq k} b_i y_i\right)$$

Now, there may be two cases arising from the value of a_k,

$$\begin{aligned}
& \Pr(x \text{ wins} \,|\, x_k = 1, y_k = 0) \\
=\ & \Pr(x \text{ wins} \,|\, x_k = 1, y_k = 0, a_k = 0) \Pr(a_k = 0) \\
+\ & \Pr(x \text{ wins} \,|\, x_k = 1, y_k = 0, a_k = 1) \Pr(a_k = 1)
\end{aligned}$$

$$\geq \frac{1}{2}(1-d) + \Pr\left(\sum_{i\neq k} a_i x_i + 1 > \sum_{i\neq k} b_i y_i\right) d$$

Now,

$$\begin{aligned}
& \Pr\left(\sum_{i\neq k} a_i x_i + 1 > \sum_{i\neq k} b_i y_i\right) \\
= \;& \Pr\left(\sum_{i\neq k} (b_i y_i - a_i x_i) < 1\right) \\
= \;& \Pr\left(\sum_{i\neq k} (b_i y_i - a_i x_i) = 0\right) + \Pr\left(\sum_{i\neq k} (b_i y_i - a_i x_i) < 0\right) \\
= \;& \Pr\left(\sum_{i\neq k} (b_i y_i - a_i x_i) = 0\right) + \frac{1}{2} - \frac{1}{2}\Pr\left(\sum_{i\neq k} (b_i y_i - a_i x_i) = 0\right) \\
= \;& \frac{1}{2} + \frac{1}{2}\Pr\left(\sum_{i\neq k} (b_i y_i - a_i x_i) = 0\right) \\
= \;& \left(\frac{1}{2} + \frac{1}{2}\sum_{j=0}^{n-1} \binom{n-1}{j}^2 \left(\frac{1}{2}d\right)^{2j} \left(1 - \frac{1}{2}d\right)^{2n-2-2j}\right) \\
= \;& \left(\frac{1}{2} + \frac{1}{2}\sum_{j=0}^{n-1} \binom{n-1}{j}^2 w^{2j} \left(1 - w\right)^{2n-2-2j}\right) \quad \text{(say)}
\end{aligned}$$

Since the above summation takes the form of the well-known Bernstein polynomial, using a conjecture on the squared Bernstein polynomials (found in Conjecture 1 in [50] and Lemma 3.5 in [56]), we can obtain the following inequality,

$$\begin{aligned}
&\geq \frac{1}{2} + \frac{1}{2^{2n-1}}\binom{2n-2}{n-1} \\
&\geq \frac{1}{2} + \frac{1}{4\sqrt{n}}
\end{aligned}$$

The above bound is obtained using the work by Stanica [110]. Then we can obtain the required probability of having a one in the bit position k, as follows,

$$\begin{aligned}\Pr(z_k = 1) \geq\ & \frac{1}{4} + \frac{1}{2}\left(\frac{1}{2}(1-d) + d\left(\frac{1}{2} + \frac{1}{4\sqrt{n}}\right)\right) \\ =\ & \frac{1}{2} + \frac{d}{8\sqrt{n}}\end{aligned}$$

The rest of the proof follows as before. $\square$

7.5.4 Fitness Evaluation Based on Subset of Bits

In several machine learning scenarios, the learner does not have access to all the data attributes. In these intrinsic noisy cases, where partial information regarding the data is available, noise handling techniques are required. In this subsection, a similar scenario is considered, where the OneMax function evaluation is dependent on a randomly chosen set of s bits of a binary string of length n.

Let S be the set of randomly chosen s bits, i.e., $S \subset [n]$ where $[n] = \{1, 2, \ldots, n\}$. Then, the noisy fitness evaluation is defined as,

$$f^{noisy}(x) = f(x_S), \text{where } x_S \subset x$$

Lemma 6. *Let $x, y \in \{0,1\}^n$ be uniformly at random chosen strings. The binary tournament selection is based on s randomly chosen bits. Let the binary tournament winner, z, be decided according to the* OneMax*. For any $k \in \{1, \ldots, n\}$, the probability that there will be a one at position k is given by*

$$\frac{1}{2} + \frac{\sqrt{s}}{8n}$$

Proof. When s random bits are sampled, and the tournament winner z is decided accordingly, the probability z will have a one in each of the s bits is at least, (From

Lemma 1),

$$\frac{1}{2} + \frac{1}{8\sqrt{s}}$$

Let S be the set of the s bits that were sampled. Then, the probability that the tournament winner has a one in position k is at least,

$$\begin{aligned}
\Pr(z_k = 1) &= \Pr(z_k = 1 \mid k \in S) \Pr(k \in S) \\
&+ \Pr(z_k = 1 | k \notin S) \Pr(k \notin S) \\
&\geq \left(\frac{1}{2} + \frac{1}{8\sqrt{s}}\right) \frac{s}{n} + \frac{1}{2}\left(1 - \frac{s}{n}\right)
\end{aligned}$$

□

Theorem 8. *If* $\mu = 32(c+1)\frac{n^2}{s}\log n$*, then the Voting algorithm correctly solves* OneMax *with probability greater than* $1 - 1/n^c$*, where, the tournament selection is performed with respect to* s *randomly sampled bits,.*

Proof. The proof of the above theorem follows as before, following the same arguments with the help of union bound and Hoeffding's inequality. It is to be noted that when all the bits are known during the tournament selection, i.e. $s = n$, the Voting algorithm requires $O(n \log n)$ function evaluations. □

7.5.5 Fitness Comparison Oracle Lies with a Probability

The fitness value comparison queried to an oracle during the selection mechanism may be noisy. This kind of uncertainty can be experienced in clustering scenarios [2]. Let the comparison oracle $\mathcal{O}$, that provides the string that has a better fitness between two strings x and y, tells the truth with a probability $(1 - l)$. Let us assume, that x has a higher fitness value than y, then the noisy fitness comparison may be defined as follows,

$$\mathcal{O}^{noisy}(x,y) = \begin{cases} x & \text{with probability } (1-l) \\ y & \text{with probability } l \end{cases}$$

Theorem 9. *Let $x, y \in \{0,1\}^n$ be uniformly at random chosen strings. Let the binary tournament winner z be decided according to the* ONEMAX *function, subject to the oracle returning the incorrect answer with probability $l < 1/7$. That is, if $|x|_1 > |y|_1$ then*

$$z = \begin{cases} x & \textit{with probability } 1-l \\ y & \textit{with probability } l \end{cases}$$

Then the probability that there will be a one at position k of the binary tournament winner is at least,

$$\frac{1}{2} + \frac{1}{8\sqrt{n}}(1-7l)$$

and the Voting algorithm solves the ONEMAX *problem in the presence of noisy comparison oracle with a high probability in $O\left(\frac{1}{(1-7l)^2} n \ln n\right)$ function evaluations.*

Proof. As in the previous analyses, with the use of theorem of total probability and Bayes' theorem, the probability that the tournament winner has a one in position k is given by,

$$\begin{aligned}
\Pr(z_k = 1) = \; & \Pr(x_k = 1 \,|\, x \text{ wins}) \\
= \; & \Pr(x_k = 1 \,|\, x \text{ wins}, \text{oracle correct})(1-l) \\
& + \; \Pr(x_k = 1 \,|\, x \text{ wins}, \text{oracle incorrect})l \\
= \; & \Pr(x \text{ wins} \,|\, \text{oracle correct}, x_k = 1)(1-l) \\
& + \Pr(x \text{ wins} \,|\, \text{oracle incorrect}, x_k = 1)l
\end{aligned}$$

Now,

$$
\begin{aligned}
&\Pr(x \text{ wins} \,|\, \text{oracle correct}, x_k = 1) \\
&= \Pr(x \text{ wins} \,|\, \text{oracle correct}, x_k = 1, y_k = 1) \Pr(y_k = 1) \\
&+ \Pr(x \text{ wins} \,|\, \text{oracle correct}, x_k = 1, y_k = 0) \Pr(y_k = 0) \\
&\geq \frac{1}{4} + \frac{1}{2} \Pr\left(\sum_{i \neq k} x_i + 1 > \sum_{i \neq k} y_i \right) \\
&= \frac{1}{4} + \frac{1}{2} \left(\frac{1}{2} + \frac{1}{4\sqrt{n}} \right) \\
&= \frac{1}{2} + \frac{1}{8\sqrt{n}}
\end{aligned}
$$

Again,

$$
\begin{aligned}
&\Pr(x \text{ wins} \,|\, \text{oracle incorrect}, x_k = 1) \\
&= \Pr(x \text{ wins} \,|\, \text{oracle incorrect}, x_k = 1, y_k = 1) \Pr(y_k = 1) \\
&+ \ \Pr(x \text{ wins} \,|\, \text{oracle incorrect}, x_k = 1, y_k = 0) \Pr(y_k = 0) \\
&\geq \frac{1}{4} + \frac{1}{2} \Pr\left(\sum_{i \neq k} (y_i - x_i) > 1 \right) \\
&= \frac{1}{4} + \frac{1}{2} \Pr\left(\sum_{i \neq k} (y_i - x_i) \geq 0 \right) - \frac{1}{2} \Pr\left(\sum_{i \neq k} (y_i - x_i) = 0 \right) - \frac{1}{2} \Pr\left(\sum_{i \neq k} (y_i - x_i) = 1 \right) \\
&\geq \frac{1}{4} + \frac{1}{2} \left[\frac{1}{2} + \frac{1}{2} \Pr\left(\sum_{i \neq k} (y_i - x_i) = 0 \right) \right] \\
&- \frac{1}{2} \Pr\left(\sum_{i \neq k} (y_i - x_i) = 0 \right) - \frac{1}{2} \Pr\left(\sum_{i \neq k} (y_i - x_i) = 0 \right) \\
&= \frac{1}{4} + \frac{1}{2} \left[\frac{1}{2} - \frac{3}{2} \Pr\left(\sum_{i \neq k} (y_i - x_i) = 0 \right) \right] \\
&= \frac{1}{4} + \frac{1}{2} \left(\frac{1}{2} - \frac{3}{2\sqrt{n}} \right) \\
&= \frac{1}{2} - \frac{3}{4\sqrt{n}}
\end{aligned}
$$

where we have used

$$\begin{aligned}
\Pr\left(\sum_{i\neq k}(y_i - x_i) = 0\right) &= \sum_{j=0}^{n-1} \Pr\left(\sum_{i\neq k} y_k = j\right) \Pr\left(\sum_{i\neq k} x_k = j\right) \\
&= \frac{1}{2^{2n-2}} \sum_{j=0}^{n-1} \binom{n-1}{j}^2 \\
&= \frac{1}{2^{2n-2}} \binom{2n-2}{n-1} \\
&\leq \frac{1}{\sqrt{\pi}\sqrt{n-1}} \\
&\leq \frac{1}{\sqrt{n}}
\end{aligned}$$

Then $\Pr(z_k = 1)$ becomes,

$$\begin{aligned}
\Pr(z_k = 1) = &\ \Pr(x_k = 1 \,|\, x \text{ wins}) \\
= &\ \left(\frac{1}{2} + \frac{1}{8\sqrt{n}}\right)(1-l) + \left(\frac{1}{2} - \frac{3}{4\sqrt{n}}\right) l \\
= &\ \frac{1}{2} + \frac{1}{8\sqrt{n}}(1 - 7l)
\end{aligned}$$

Now, by using the Hoeffding's inequality, the probability the the vote is incorrect for bit position k is given by,

$$\begin{aligned}
\Pr\left(p_k \leq \frac{\mu}{2}\right) = &\ \Pr\left(p_k \leq \left(p\mu - p\mu + \frac{\mu}{2}\right)\right) \\
\leq &\ \exp\left(-2\left(p - \frac{1}{2}\right)^2 \mu\right) \\
= &\ \exp\left(-2\mu\frac{1}{64n}(1-7l)^2\right) = \frac{1}{n^{c+1}}
\end{aligned}$$

The rest of the proof follows as the previous analyses. □

7.6 Voting Algorithm on Linear and Monotone Functions

As the Voting algorithm samples strings with close to $n/2$ ones, it is unlikely to efficiently solve all linear functions. In the case of BinVal the selection will be dominated by the highest order bits, and the voting on the low order bits will essentially be random. Nevertheless, it is interesting to see for what linear functions the algorithm remains efficient (in the sense of having a polynomial runtime).

7.6.1 Non-noisy Monotone and Linear Functions

We look at the broader class of *monotonic* functions which have the property that the fitness always increases when a zero bit is changed to a one [31].

Denoting by $\mathbf{e_k}$ the binary string with a one in position k and zeros elsewhere, then we have for integer valued monotonic functions, f:

$$x_k = 1 \Longrightarrow f(x) \geq f(x \oplus \mathbf{e_k}) + 1$$

for all strings x.

Theorem 10. *Let f be a monotonic function. Then the Voting algorithm optimises f in $O(|\operatorname{Im} f|^2 \log n)$ function evaluations.*

Proof. As with the case of OneMax, the probability that the winner of a tournament between two random strings x and y has a one in position k is equal to,

$$\begin{aligned}&\Pr(x_k = 1 \,|\, x \text{ wins})\\&= \frac{1}{2}\Pr(x \text{ wins} \,|\, x_k = 1, y_k = 1) + \frac{1}{2}\Pr(x \text{ wins} \,|\, x_k = 1, y_k = 0)\end{aligned}$$

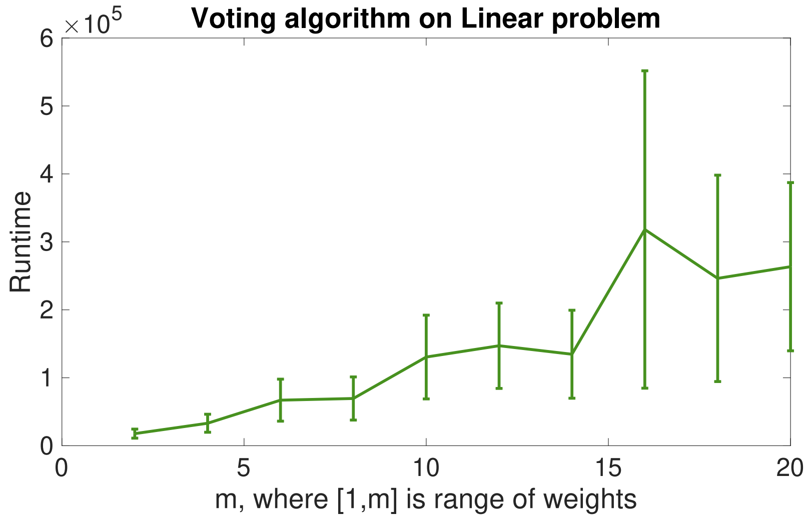

Figure 7.1: Runtime of the Voting algorithm on linear functions, with weights chosen randomly from the range $1, \ldots, m$.

$$\geq \frac{1}{4} + \frac{1}{2}\Pr(f(x) > f(y) \,|\, x_k = 1, y_k = 0)$$
$$\geq \frac{1}{4} + \frac{1}{2}\Pr(f(x \oplus \mathbf{e_k}) + 1 > f(y) \,|\, x_k = 1, y_k = 0)$$

(where we consider a worst case scenario that the bit in question gains only an increase of one in the fitness of x)

$$= \frac{1}{4} + \frac{1}{2}\Pr(f(y) - f(x \oplus \mathbf{e_k}) = 0 \,|\, x_k = 1, y_k = 0)$$
$$+ \frac{1}{2}\Pr(f(y) - f(x \oplus \mathbf{e_k}) < 0 \,|\, x_k = 1, y_k = 0)$$
$$= \frac{1}{2} + \frac{1}{4}\Pr(f(y) - f(x \oplus \mathbf{e_k}) = 0 \,|\, x_k = 1, y_k = 0)$$

(by symmetry, since both y and $x \oplus \mathbf{e_k}$ are conditioned to have a zero in bit k)

$$= \frac{1}{2} + \frac{1}{4}\Pr(f(y) = f(x \oplus \mathbf{e_k}) \,|\, x_k = 1, y_k = 0)$$
$$= \frac{1}{2} + \frac{1}{4}\sum_{\phi \in \Phi} \Pr(f(y) = \phi \,|\, y_k = 0) \Pr(f(x \oplus \mathbf{e_k}) = \phi \,|\, x_k = 1)$$

(where by Φ we denote the set of all values f can take on, conditioned on bit k of its argument having value zero)

$$
\begin{aligned}
&= \frac{1}{2} + \frac{1}{4}\sum_{\phi\in\Phi} \Pr(f(y)=\phi \,|\, y_k=0)^2 \\
&\geq \frac{1}{2} + \frac{1}{4|\Phi|} \\
&\geq \frac{1}{2} + \frac{1}{4|\operatorname{Im} f|}
\end{aligned}
$$

(using the Cauchy-Schwarz inequality[1]). The result follows as before. □

In the case of linear functions in which there is a set of positive integer weights W, and

$$f(x) = \sum_{i=1}^{n} w_i x_i$$

we have $|\operatorname{Im} f| \leq 1 + \sum_{i=1}^{n} w_i$ to give a runtime bound of

$$O((\sum_{i=1}^{n} w_i)^2 \log n) = O(\overline{w}^2 n^2 \log n)$$

where $\overline{w}$ is the average of the weights. We see that the Voting algorithm can solve linear functions in polynomial time, as long as the average of the weights is polynomial.

We can see this is an over-estimate in the case of OneMax. This is due to the use of the Cauchy-Schwarz inequality, which avoids the need for detailed combinatorial analysis, but gives a weaker bound.

To test how the runtime depends on the weights, we ran experiments with $n = 100$, choosing weights uniformly at random from the range $1, \ldots, m$. The

[1]Since, $\sum_{\phi\in\Phi} \Pr(f(y)=\phi \,|\, y_k=0) = 1$, by Cauchy-Schwarz inequality, we have

$$
\begin{aligned}
\left(\sum_{\phi\in\Phi} 1 \times \Pr(f(y)=\phi \,|\, y_k=0)\right)^2 &\leq \sum_{\phi\in\Phi} \Pr(f(y)=\phi \,|\, y_k=0)^2 \times \sum_{\phi\in\Phi} 1 \\
\Rightarrow \Pr(f(y)=\phi \,|\, y_k=0)^2 &\geq \frac{1}{|\Phi|}
\end{aligned}
$$

results, shown in figure 7.1, indicate that the dependency on the average weight is closer to linear than quadratic.

7.6.2 Dynamic BinVal problem

In order to analyse the performance of the voting algorithm in a noisy setting of linear problems, we consider the dynamic version of the BinVal problem, which is a limiting case of dynamic linear functions [74, 75]. In dynamic functions, the fitness function keeps on changing at each generation, however, the optimum remains the same.

The *Dynamic*BinVal problem is defined on $\{0,1\}^n \to \mathbb{R}$ such that,

$$\textit{Dynamic}\textsc{BinVal}(x) = \sum_{i=1}^{n} 2^{n-i} x_{\pi(i)}$$

where, $\pi(i) : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ is a permutation , i.e., the order of the bits are chosen randomly in each function evaluation.

Lengler & Riedi [75] have analysed the runtime of (μ+1) EA for the *Dynamic*BinVal problem and they show that the runtime is exponential unless the optimisation starts with string located very close to the optimum, i.e., are within the ε-neighbourhood of the optimum. It is, however, clear from our following result that voting algorithm can solve the *Dynamic*BinVal problem with uniformly random string initialisation in polynomial runtime, $O(n^2 \log n)$.

Theorem 11. *The voting algorithm correctly solves the Dynamic*BinVal *with high probability in $O(n^2 \log n)$ function evaluations.*

Proof.

$$\begin{aligned}
&\Pr(z_k = 1) \\
&= \Pr(z_k = 1 \mid k \text{ is leading bit}) \Pr(k \text{ is leading bit})
\end{aligned}$$

$$
\begin{aligned}
&+ \Pr(z_k = 1 \,|\, k \text{ is not leading bit}) \Pr(k \text{ is not leading bit}) \\
&= 1 \times \frac{1}{n} + \frac{1}{2} \times \left(1 - \frac{1}{n}\right) = \frac{1}{2} + \frac{1}{2n}
\end{aligned}
$$

Following the next steps as in the previous analyses, we may conclude that $O(n^2 \log n)$ function evaluations are needed to solve *Dynamic*BinVal correctly using the voting algorithm with a high probability. □

7.7 Voting Algorithm for LeadingOnes Problem

The observation that the Voting algorithm always samples strings with close to $n/2$ ones, makes it clear that it cannot efficiently solve the LeadingOnes problem.

$$
\textsc{LeadingOnes}(x) = \sum_{i=1}^{n} \prod_{j=1}^{i} x_i
$$

Algorithm 19: The Significant Bit Voting Algorithm

```
Let z = (−1, ..., −1);
repeat n times times
    Let pop be the empty population;
    repeat μ times times
        for 1 ≤ i ≤ n do
            if z_i = −1 then
                x_i = 0 or 1 uniformly at random ;
            else
                x_i = z_i;
            end
        end
        Add x to pop;
    end
    Rank pop by fitness, and keep only the best μ/3 strings;
    Let v be the sum of all strings in pop;
    Let k be the index for which z_k = −1 and |v_k − μ/6| is maximised;
    Let z_k = [v_k > μ/6]
end
Return z;
```

However, we can still use the voting idea to create a reasonably efficient algorithm. We do this by voting for one bit at a time, starting from the most significant bit. We also make use of truncation selection, rather than tournament selection. This variant is shown as Algorithm 19.

The vector z keeps track of which bits have been determined. At each iteration, μ strings are generated using the predetermined bits recorded in z, and generating the remainder randomly. The best third of these are kept, and we determine which bit (that has not already been set) has the largest vote. The value of this one is then set by the vote and recorded in z.

This algorithm runs in μn time, and works by determining the bit values of the final solution one at a time. We will set $\mu = (c+2)\log n$. When applied to LeadingOnes a number of things could go wrong. Firstly, not enough strings in the top third of the population might have the next correct bit value. However, since the probability of generating the correct bit in the population is 1/2, the probability that the top third does not contain only the correct next bit value is $O(1/n^{c+2})$. It might also happen by chance that one of the other bit positions also gets only one value represented in the top third of the population. Again, the probability that this happens is $O(1/n^{c+2})$. There are at most n things that could go wrong in each iteration, which means the probability of getting an incorrect bit in an iteration is $O(1/n^{c+1})$. Therefore, by the union bound, the probability of not returning the correct solution is $O(1/n^{c})$. We can thus show the following,

Theorem 12. *The Significant Bit Voting algorithm solves* LeadingOnes *(with high probability) in $O(n \log n)$ function evaluations.*

It should be noted that this result depends on the fine balance between population size and selection pressure, which needs to ensure we get the leading bit right, but is very unlikely to accidentally set an incorrect bit value.

7.8 Experimental Analysis of Voting Mechanism

We now consider the use of voting as a practical method to enhance the performance of population based algorithms.

Algorithm 20: Voting UMDA

Set $t \leftarrow 0$;
Let $p(0) = (0.5, \ldots, 0.5)$;
Sample $p(t)$ λ times to form population;
while *termination condition not reached* **do**
  Set $t \leftarrow t + 1$;
  Let $q^1, q^2, \ldots, q^\lambda$ be the population sorted according to fitness;
  for $i = \{1, \ldots, n\}$ **do**
    Let $X_i = \sum_{j=1}^{\mu} q_i^j$;
    Set $p(t)_i = \frac{X_i}{\mu}$;
    if $p(t)_i < 1/n$ **then**
      Set $p(t)_i = 1/n$;
    end
    if $p(t)_i > 1 - 1/n$ **then**
      Set $p(t)_i = 1 - 1/n$;
    end
    if $p(t)_i = 0.5$ **then**
      Set v_i to be 0 or 1 at random;
    end
    if $p(t)_i \neq 0.5$ **then**
      Set $v_i = [p(t) > 0.5]$;
    end
  end
  Report vote v;
end

The idea is that the population may be "pointing" to the correct solution long before it actually converges on it. For example, for UMDA, it is possible that the bit frequencies are all in the correct direction before it has produced a single copy of the optimum.

To explore this idea, we look at three different algorithms (UMDA, PCEA and cGA) and empirically study the vote of the population at each iteration. For the purposes of these experiments, the vote has no effect on the running of the algorithm; we simply report the fitness of the population vote at each iteration. The algorithm designs with voting are illustrated in Algorithms 20, 21, and 22.

Algorithm 21: Voting PCEA

```
Initialise a random population of λ strings;
while termination condition not reached do
    repeat λ times
        Choose parents X^α and X^β uniformly from population;
        Generate a random vector a = {a_1, a_2, ..., a_n} ∈ {0,1}^n;
        Create complementary children X^μ and \overline{X^μ} by uniform crossover
          such that
                X_i^μ = a_i X_i^α + (1 − a_i) X_i^β
                \overline{X_i^μ} = (1 − a_i) X_i^α + a_i X_i^β
        The better of the offspring goes to the next generation;
    end
    for i = {1, ..., n} do
        Let X_i = Σ_{j=1}^{λ} q_i^j, where q^1, ..., q^λ are members of next population;
        Set p(t)_i = X_i / λ;
        if p(t)_i = 0.5 then
            Set v_i = [p(t) > 0.5];
        end
        if p(t)_i ≠ 0.5 then
            Set v_i = [p(t) > 0.5];
        end
    end
    Report vote v;
end
```

For UMDA [81], we used truncation selection to pick the best $\lambda/2$ strings at each iteration. For PCEA we used tourmanent selection between each pair of generated offspring as described in [91]. For cGA, the parameter K represents the population size, although only two individuals are created at each iteration [58]. The vote is performed by looking at the bit probabilities.

Algorithm 22: Voting cGA

Set $t \leftarrow 0$;
Let $p(0) = (0.5, \ldots, 0.5)$;
while *termination condition not reached* **do**
  for $i \in \{1, 2, \ldots, n\}$ **do**
    Set $x_i = 1$ with probability $p(t)_i$ and $x_i = 0$ with probability $1 - p(t)_i$;
    Set $y_i = 1$ with probability $p(t)_i$ and $y_i = 0$ with probability $1 - p(t)_i$;
  end
  if $f(x) < f(y)$ **then**
    Swap x and y;
  end
  for $i \in \{1, 2, \ldots, n\}$ **do**
    if $x_i > y_i$ **then**
      Set $p(t+1)_i = p(t)_i + \frac{1}{K}$;
    end
    if $x_i < y_i$ **then**
      Set $p(t+1)_i = p(t)_i - \frac{1}{K}$;
    end
    if $x_i = y_i$ **then**
      Set $p(t+1)_i = p(t)_i$;
    end
  end
  Report vote v;
  Set $t \leftarrow t + 1$;
end

7.8.1 Experiments on OneMax

For each experiment, we fixed $n = 200$ and examined the runtime (in terms of number of function evaluations) of the algorithms for different population sizes. The results on OneMax (without any added noise) are shown for UMDA in figure 7.2; for PCEA in figure 7.3; and for cGA in figure 7.4.

It can be seen in all cases, that the vote improves the performance for larger population sizes, with the effect being significant for all the considered algorithms. In each plot, we show error bars of one standard-deviation. Each relevant comparison has been tested using the Mann-Whitney test, and found to be significant at the 95% level.

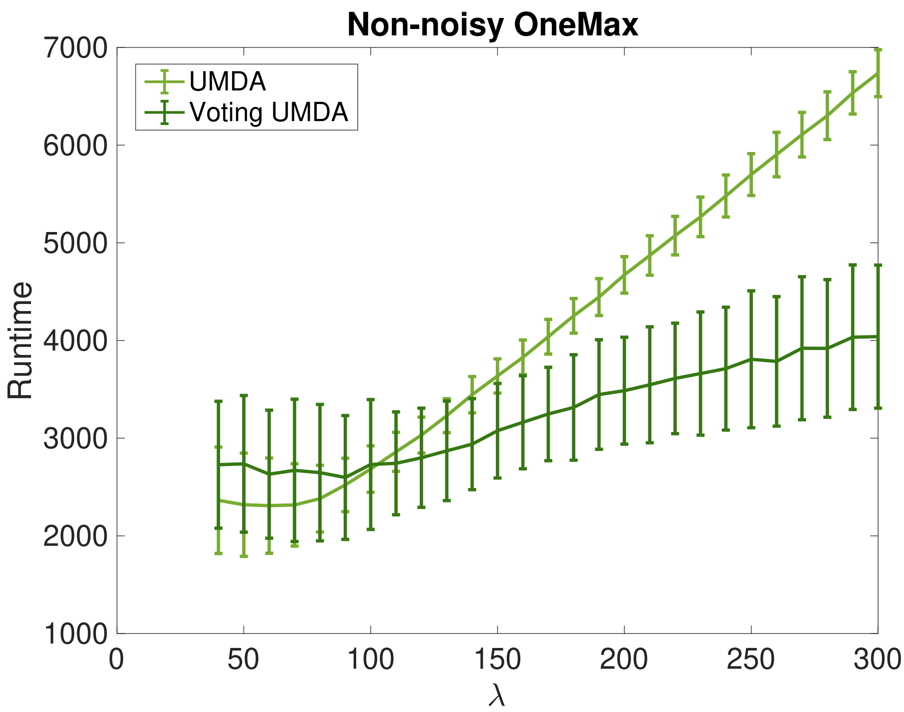

Figure 7.2: Comparison of UMDA and Voting UMDA while solving the (non-noisy) OneMax function

The voting mechanism is beneficial for these algorithm because, when the population is small, it often happens that some of the bit probabilities will go the wrong value, where they meet the lower margin (set to $1/n$ in all cases). By the time these have recovered, the rest of the bit values are at the upper margin (set to $1-1/n$). When all except one or two bits are at the upper margin, and the remaining bits just below $1/2$, it is likely that the optimum solution will be produced, even though the vote will be incorrect. When the population sizes are larger, this does not happen, and each bit probability quickly exceeds $1/2$, giving the correct vote.

The runtime is analysed for UMDA and PCEA while population size λ is varied. Similarly, the parameter K in cGA is varied to analyse how it affects the runtime. The voting algorithm solves the non-noisy OneMax problem on an average over

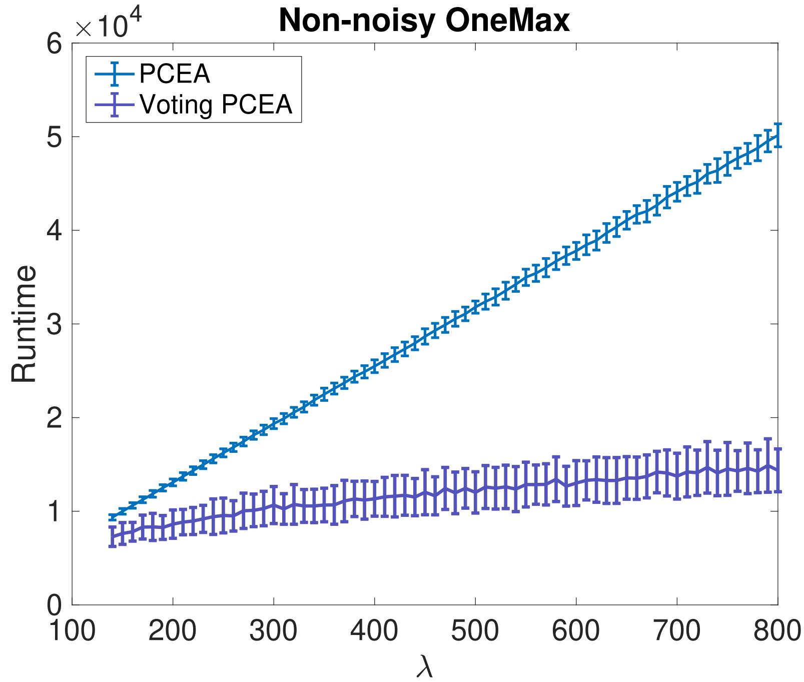

Figure 7.3: Comparison of PCEA and Voting PCEA while solving the (non-noisy) OneMax function

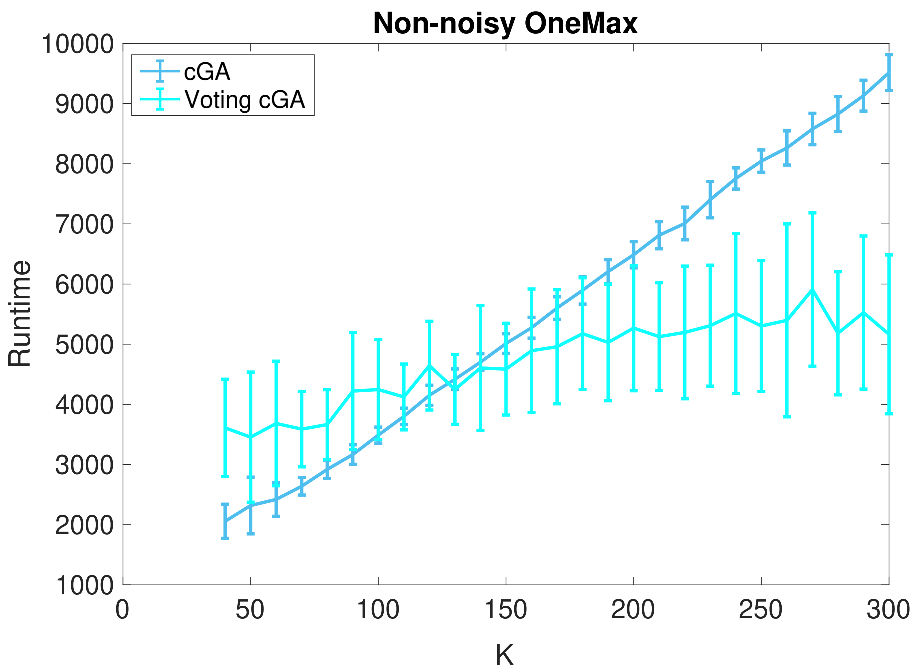

Figure 7.4: Comparison of cGA and Voting cGA while solving the (non-noisy) OneMax function

100 runs, in 13110 function evaluations.

7.8.2 Experiments on OneMax with Posterior Noise

For the experiments with noisy OneMax for $n = 200$, we use Gaussian noise with $\sigma = 5$. The results are shown for UMDA in figure 7.5; for PCEA in figure 7.6; and for cGA in figure 7.7.

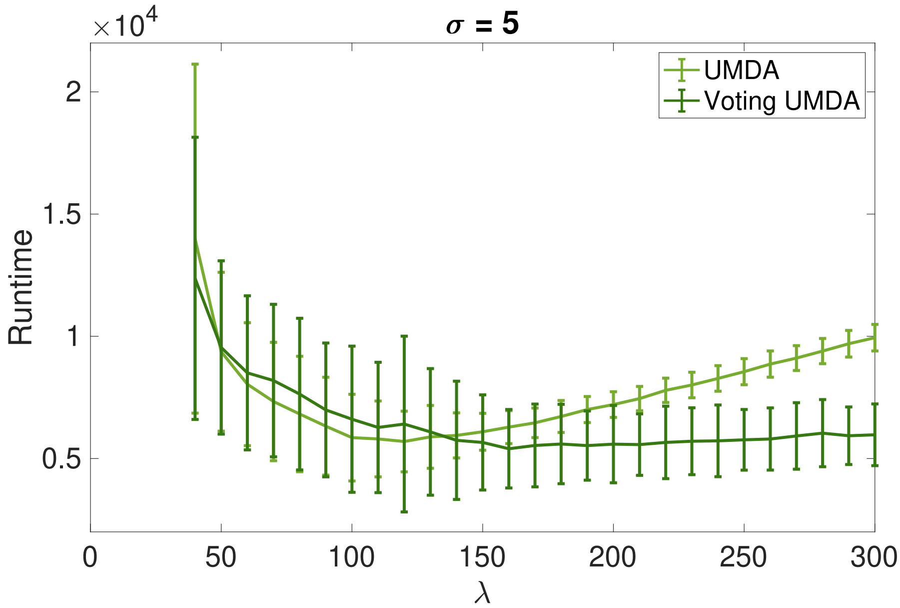

Figure 7.5: Comparison of UMDA and Voting UMDA while solving the noisy OneMax function

Again, we see that the voting improves the algorithms for large population sizes. For UMDA with a small population, there is little difference with and without voting.

The voting algorithm solves the noisy OneMax with $\sigma = 5$ problem on an average over 100 runs, in 17943 function evaluations.

7.8.3 Experiments on Non-noisy Linear

For the non-noisy Linear problem for $n = 200$, the results are illustrated in Figure 7.8 for UMDA; Figure 7.9 for cGA and Figure 7.10 for PCEA. Random

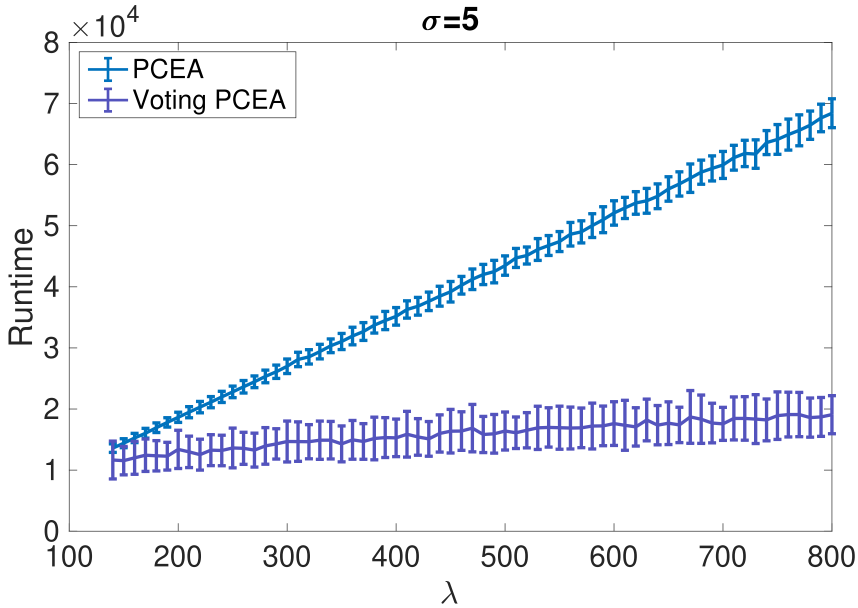

Figure 7.6: Comparison of PCEA and Voting PCEA while solving the noisy OneMax function

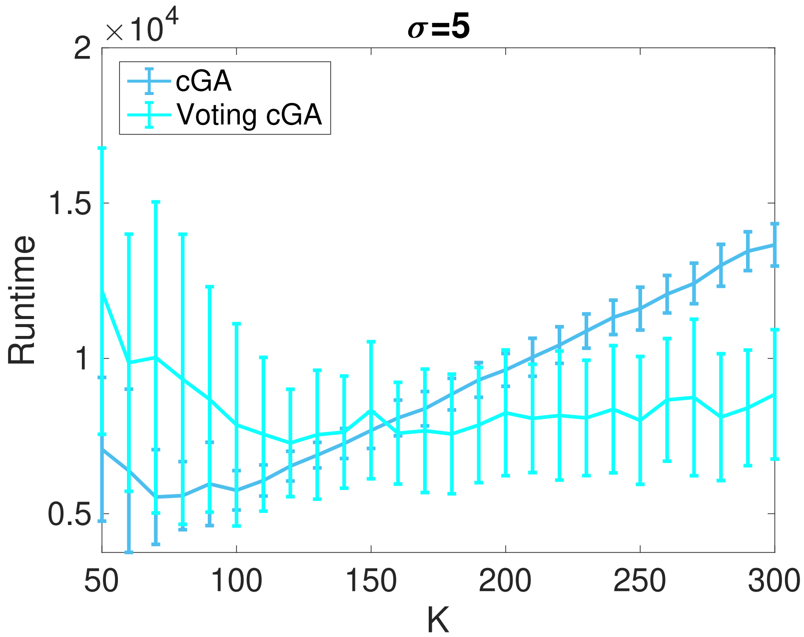

Figure 7.7: Comparison of cGA and Voting cGA while solving the noisy OneMax function

problem instances are chosen with weights ranging from $1, \ldots, m$ with m varying from 1 to 20.

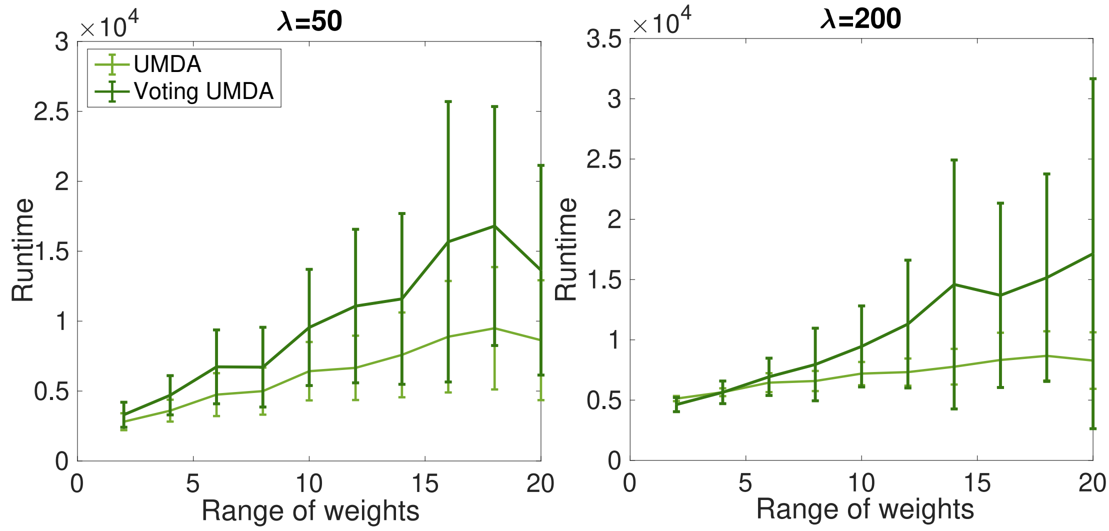

Figure 7.8: Comparison of UMDA and Voting UMDA while solving the non-noisy LINEAR function

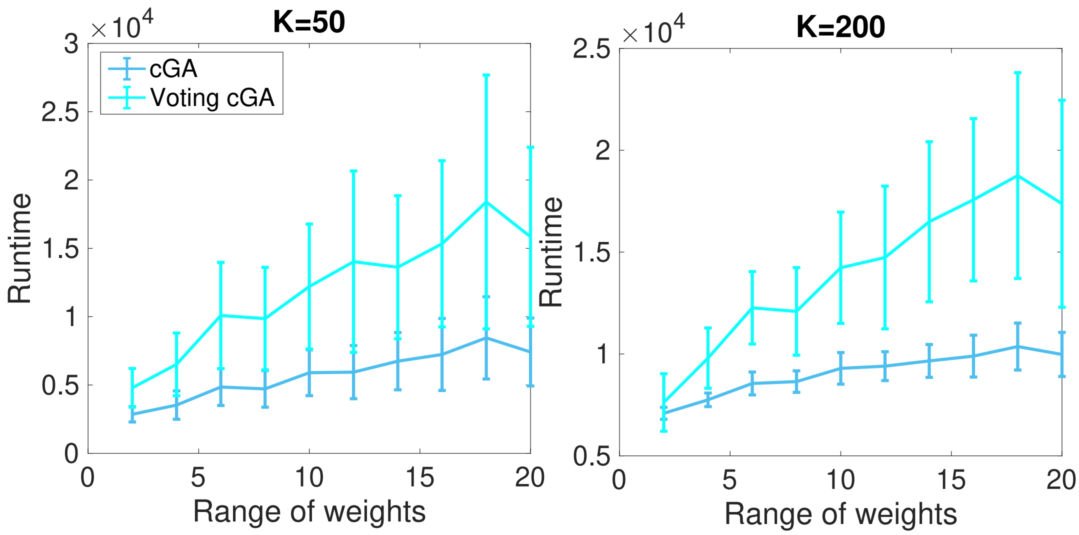

Figure 7.9: Comparison of cGA and Voting cGA while solving the non-noisy LINEAR function

For UMDA, population sizes $\lambda = 50$ and 200 are chosen, representing the regimes where voting does not and does help in ONEMAX respectively. Similarly, for cGA the different regimes are analysed by considering $K = 50$ and 200. The algorithms are able to find the optimum in each cases.However, we see that the voting mechanism does not help as the weights get larger, which is consistent with our analysis of the Voting algorithm on such problems.

However, for such small values of population sizes, the PCEA cannot solve the LINEAR problem. According to the theoretical result in [91], we choose the population size $\lambda = 10 \times \sqrt{n} \times \log n$. Considering this choice of λ, both PCEA with and without voting solves the problems.

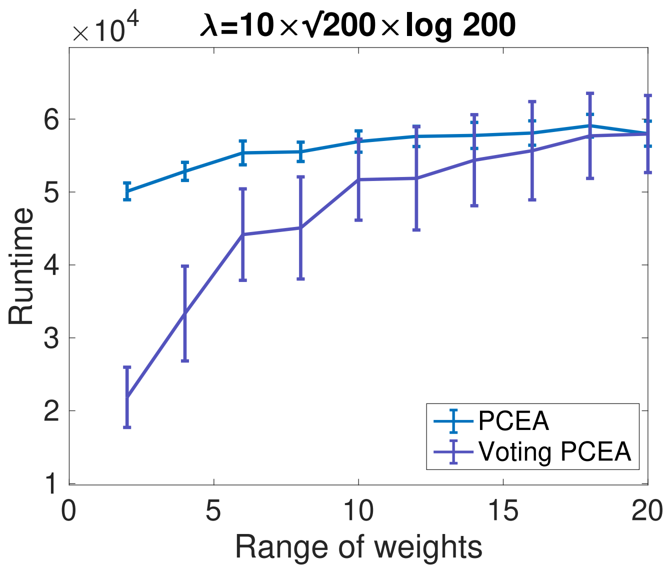

Figure 7.10: Comparison of PCEA and Voting PCEA while solving the non-noisy LINEAR function

7.9 Summary

We have studied the use of voting as a heuristic method. It is particularly effective for the noisy ONEMAX problem with different variants of noise. We prove that the upper bounds on the runtime of ONEMAX with posterior and prior noise are better than any other algorithm we are aware of. In case of partial evaluation of fitness functions, as well, the voting algorithm would require significantly lesser function evaluations than the existing literature. We also analyse the runtime on ONEMAX with two other variants of noise relevant in learning and optimisation problems. A variant of the voting idea which works one bit at a time is reasonably efficient for LeadingOnes. The voting approach works less well for general linear problems, and we have investigated this effect, in fact showing an upper bound for general monotonic functions. Finally, we have empirically studied the idea of incorporating voting into a population-based algorithm and conclude that this may be effective for large population sizes.

Chapter 8

Conclusions & Future Scope of Research

The main aim of this thesis is to determine efficient evolutionary optimisation approaches to handle noisy combinatorial problems with large levels of noise. The optimisation problem can range from being a simple linear problem to having constraints and/or with multiple objectives. Several noise models relevant in optimisation and learning have been investigated in this context.

Firstly, this work has explored the existing empirical and theoretical literature on noise handling for combinatorial problems, while reviewing the very early research on noise handling with EAs, as well as the recent theoretical runtime analyses of EAs in Chapter 2. The review of the early literature which were mostly empirical in terms of algorithm convergence, provided some important insights regarding noise handling with EAs. Since early research in noisy optimisation, one of the most common noise handling techniques is sampling the fitness of candidate solution multiple times to estimate the noise-free fitness. Population-based EAs have been also been shown to be helpful in noisy optimisation. A debate on whether mutation or crossover is favourable has also been a hot topic over many years.

Recent theoretical research on EAs have focused on estimating the expected runtime of EAs, more generally, estimating the average number of function evaluations that are required to reach the optimum. In the context of noisy combinatorial optimisation, the efficiency of resampling, the mutation-based hill climber $(1+1)$–EA, some mutation-only algorithms with a population, a crossover-based algorithm, Paired Crossover Evolutionary Algorithm (PCEA) with no mutation and some estimation of distribution algorithms (EDAs) have been investigated. Although these algorithms have polynomial runtimes in many cases, PCEA (employing binary tournament selection and uniform crossover) provides a better asymptotic runtime bound than the remaining ones.

Building upon the existing literature, a range of empirical studies has been carried out to gain insights regarding the best choice of an EA and its noise handling capacity in practice. A wide array of EAs has been chosen carefully for empirical analysis of their performances on simple combinatorial problems, OneMax and Linear with noisy fitness evaluations within a fixed runtime budget in Chapter 3. This array of algorithms consists of mutation-based hill climber ($(1+1)$–EA), a mutation-only population-based algorithm, crossover-only population-based algorithm (PCEA) and some estimation of distribution algorithms, without (cGA) and with an explicit population (UMDA and PBIL). In these experiments, the noise is considered as additive Gaussian with large noise variances. The observations from these empirical analyses are as follows,

- (1+1)–EA is capable to cope with only the smallest levels of noise, as expected from the theoretical analyses.
- The mutation-population algorithm and cGA have theoretical polynomial runtime for noisy OneMax, but fail to be useful in practice compared to the other algorithms.
- The performance of PBIL is very much similar to cGA. They are not able to cope with even these small levels of noise within the given fixed budget of

function evaluations.

- The PCEA algorithm handles noise well on the simple test problems with significant noise variances.

- Interestingly, UMDA also handles these cases well, with even a slightly better performance than PCEA in some cases. This may be due to the fact that UMDA has a stronger selection method (truncation selection) than PCEA (which uses a tournament on pairs of offspring). It also should be noted that UMDA employs a mechanism similar to *genepool crossover*, where at each bit position, the offspring bit is obtained by recombination of that bit across the whole parent population. It is therefore hypothesised that PCEA and UMDA are highly similar in operation.

These experiments to find the better performing algorithms, provided a notion regarding the choice of efficient algorithm for harder combinatorial problems (that have a 'packing' structure which might make them amenable to algorithms that are capable to solve noisy OneMax efficiently), the SubsetSum, Knapsack and SetCover problems. The performance of PCEA and UMDA on these problems are studied in Chapter 4. For the SubsetSum problem, the presence of noise is considered in the fitness function evaluations. In case of the Knapsack problem, two variants of noise are considered – the first version with presence of posterior additive noise in fitness evaluation and the second version is more complex with presence of noise in the judgement with respect to the weights. In solving the SetCover problem, two different representations of the fitness function are considered - the first one with constraints and the second one with a penalty function. In each of the representations, posterior additive noise is considered. A range of forty problem instance with an array of noise variances are studied. It is observed that,

- Both PCEA and UMDA are can handle noise well in these problem instances, with UMDA being slightly better as confirmed with Mann-Whitney U test. The recombination style operations make them particularly efficient in handling complex combinatorial instances in presence of large noise variances.

At this stage we are interested to know if a multi-objective representation of the SetCover problem might be more efficient to handle instead of the single-objective versions, as illustrated in some recent studies analysing single and multi-objective optimisation problems. Published results are not known on multi-objective combinatorial optimisation with noisy function evaluations. To begin with the experiments on multi-objective problems, we start with the toy problem CountingOnesCountingZeroes (COCZ) problem in Chapter 5. Several variants of multi-objective UMDA owing to its successful performance in solving single-objective noisy combinatorial problems are studied. The multi-objective version of the hill climber SEMO and the well-studied NSGA–II are also considered to compare the performance of moUMDA with them. The performance indicator, hypervolume is used to compare the performances of the multi-objective algorithms which provides an analysis of the spread of the non-dominated solutions found and the proximity to the Pareto front, in a reasonable function evaluations budget.

For solving multi-objective problems, it needs to be ensured that diversity is preserved in the population, so several diversification techniques have been implemented in the algorithms. A simple diversity preservation technique by avoiding duplicates in the population is studied. Clustering mechanisms (K-means and Hierarchical Agglomerative Clustering) to ensure that the candidate solutions are spread out across the feasible region and the diversity is maintained in the population. Another diversification technique is considered based on driving the search towards those non-dominated solutions that increase the Hypervolume performance indicator of the population. The consideration of an archive of non-dominated solutions and creating the next population on the basis of it is also analysed. The observations from these experimental results on COCZ are as follows,

- SEMO loses population diversity in very small levels of noise. The mutation operator is not helpful in preserving the diversity in noisy multi-objective combinatorial instances.

- The plain moUMDA and the version forbidding duplicates in the population both have the curious property that their performance improves with the presence of low levels of noise, and then degrade at higher levels of noise. However, moUMDA with no duplicates has the best performance in large variances of noise.

- In the presence of noise, the archive would contain dominated solutions along with the non-dominated solutions, which is unhelpful in the optimisation process if the archive is used to generate the next generation of population.

- The performance of rest of the algorithms including NSGA–II degrades with noise variance.

- Clustering helps in diversity preservation and both of the clustering mechanisms have similar performance.

In the next stage, the experiments on the multi-objective SetCover show the following,

- moUMDA allowing no duplicates handles high standard deviations of noise significantly better than other algorithms (moUMDA employing K-means and NSGA–II).

- The performance of NSGA–II becomes worse as the standard deviation of noise increases and the problem size increases.

- It is very interesting to note that, better feasible solutions are found while solving the multi-objective version of the same noisy SetCover problem having the constrained or penalty function approach.

In Chapter 6, as a real-world application of these analyses on multi-objective combinatorial problems, the path planning for a ground surveillance mission by an Unamanned Aerial Vehicle (UAV) is considered. An uncertain scenario is considered such that due to uncertain weather factors such as clouds, fog and haze, the visibility of the UAV is obscured with a random noise. Two variants of multi-objective UMDA with and without K-Means clustering are proposed and compared with respect to the performance of the well-studied NSGA–II adapted to this UAV problem. It is observed that,

- The archive of non-dominated solutions from the moUMDA variants have a higher hypervolume than NSGA–II.
- In terms of ground coverage percentage in presence of random noise arising from weather factors, moUMDA variants are a better choice.

In the final stage of this thesis in Chapter 7, the theoretical analysis of a newly introduced heuristic algorithm, the Voting algorithm, based on bit-wise majority voting of search strings has shown state-of-the-art performance in solving the benchmark, noisy OneMax with several noise variants. The observations are as follows,

- Voting as a recombination strategy can be effective to find the optimum in simple noisy combinatorial instances in a wide variety of noise models relevant is optimisation and learning.
- Since the Voting algorithm samples strings with close to n/2 ones, it is unlikely that it will be effective for all linear function, but can solve in general monotonic functions in reasonable polynomial time.

- A version of voting algorithm can be fairly efficient for solving LeadingOnes.
- Incorporating voting into population-based algorithms can provide significant speed ups to the performance and this can be effective for large population sizes.

8.1 Future Scope of Research

The findings of this thesis contribute to the understanding that recombination can be beneficial in noisy combinatorial optimisation. To provide a conclusive observation, the experimental performances of mutation-based algorithms and the algorithms with recombination operations has been judged separately in the context of noisy combinatorial problems. Since, the theoretical analysis of recombination operators can be highly challenging, the insight regarding the choice of good algorithms to optimise a realistic noisy combinatorial problems is helpful. The performance of UMDA has been promising in practice. However, a theoretical proof of the expected runtime of UMDA in presence of large noise variance would be interesting to gain more clarity regarding its performance. A future direction of research could be enhancing the learning strategy of the probability distribution of the selected best search strings such as in case of multi-variate EDAs (like hBOA with Bayesian networks) in the context of noisy optimisation. Since voting offers significant performance speed-ups to population-based algorithms like UMDA, investigating this further with considerations of dependencies within bit positions and assignment of weights for solving more complex combinatorial problems would be an obvious further step.

Bibliography

[1] David H. Ackley. "Stochastic iterated genetic hillclimbing". PhD thesis. Carnegie Mellon University, 1987.

[2] Raghavendra Addanki, Sainyam Galhotra, and Barna Saha. "How to Design Robust Algorithms using Noisy Comparison Oracle". In: *arXiv preprint arXiv:2105.05782* (2021).

[3] Aishwaryaprajna and Jonathan E. Rowe. "Evolutionary Algorithms for Solving Unconstrained, Constrained and Multi-objective Noisy Combinatorial Optimisation Problems". In: *arXiv preprint arXiv:2110.02288* (2021).

[4] Aishwaryaprajna and Jonathan E. Rowe. "Noisy combinatorial optimisation by evolutionary algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion.* 2019, pp. 139–140.

[5] Akiko N. Aizawa and Benjamin W. Wah. "Scheduling of Genetic Algorithms in a Noisy Environment". In: *Evolutionary Computation* 2.2 (1994), pp. 97–122.

[6] Youhei Akimoto, Sandra Astete-Morales, and Olivier Teytaud. "Analysis of runtime of optimization algorithms for noisy functions over discrete codomains". In: *Theoretical Computer Science* 605 (2015), pp. 42–50. ISSN: 0304-3975.

[7] Gautham Anil and R Paul Wiegand. “Black-box search by elimination of fitness functions”. In: *Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms*. 2009, pp. 67–78.

[8] Meghna Babbar, Ashvin Lakshmikantha, and David E. Goldberg. “A modified NSGA-II to solve noisy multiobjective problems”. In: *Genetic and Evolutionary Computation Conference (GECCO 2003)*. Vol. 2723. 2003, pp. 21–27.

[9] Shumeet Baluja. *Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning.* Tech. rep. Carnegie-Mellon University, Pittsburgh PA, Department Of Computer Science, 1994.

[10] Matthieu Basseur and Eckart Zitzler. “Handling uncertainty in indicator-based multiobjective optimization”. In: *International Journal of Computational Intelligence Research* 2.3 (2006), pp. 255–272.

[11] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J. Gutjahr. “A survey on metaheuristics for stochastic combinatorial optimization”. In: *Natural Computing* 8.2 (2009), pp. 239–287.

[12] Jürgen Branke. “Creating robust solutions by means of evolutionary algorithms”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 1998, pp. 119–128.

[13] Jürgen Branke and Christian Schmidt. “Selection in the Presence of Noise”. In: *Genetic and Evolutionary Computation — GECCO 2003*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 766–777.

[14] Jürgen Branke and Christian Schmidt. “Sequential Sampling in Noisy Environments”. In: *Parallel Problem Solving from Nature - PPSN VIII*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 202–211.

[15] Jürgen Branke, Christian Schmidt, and Hartmut Schmeck. "Efficient fitness estimation in noisy environments". In: *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. 2001, pp. 243–250.

[16] Jack E. Bresenham. "Algorithm for computer control of a digital plotter". In: *IBM Systems journal* 4.1 (1965), pp. 25–30.

[17] Dimo Brockhoff, Manuel López-Ibánez, Boris Naujoks, and Günter Rudolph. "Runtime analysis of simple interactive evolutionary biobjective optimization algorithms". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2012, pp. 123–132.

[18] D Buche, Peter Stoll, Rolf Dornberger, and Petros Koumoutsakos. "Multiobjective evolutionary algorithm for the optimization of noisy combustion processes". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 32.4 (2002), pp. 460–473.

[19] Lam T. Bui, Hussein A Abbass, and Daryl Essam. "Fitness inheritance for noisy evolutionary multi-objective optimization". In: *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. 2005, pp. 779–785.

[20] Erick Cantú-Paz. "Adaptive Sampling for Noisy Problems". In: *Genetic and Evolutionary Computation – GECCO 2004*. Ed. by Kalyanmoy Deb. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 947–958.

[21] Dogan Corus, Duc-Cuong Dang, Anton V. Eremeev, and Per Kristian Lehre. "Level-based analysis of genetic algorithms and other search processes". In: *IEEE Transactions on Evolutionary Computation* 22.5 (2017), pp. 707–719.

[22] Duc-Cuong Dang and Per Kristian Lehre. "Efficient optimisation of noisy fitness functions with population-based evolutionary algorithms". In: *Pro-*

ceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII. ACM. 2015, pp. 62–68.

[23] Duc-Cuong Dang and Per Kristian Lehre. “Runtime analysis of non-elitist populations: From classical optimisation to partial information”. In: *Algorithmica* 75.3 (2016), pp. 428–461.

[24] Raphaël Dang-Nhu, Thibault Dardinier, Benjamin Doerr, Gautier Izacard, and Dorian Nogneng. “A new analysis method for evolutionary optimization of dynamic and noisy objective functions”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2018, pp. 1467–1474.

[25] Lawrence Davis. “Bit-climbing, representational bias, and test suit design”. In: *Proceedings of International Conference of Genetic Algorithm*. Springer, Heidelberg. 1991, pp. 18–23.

[26] Kenneth Alan De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. University of Michigan, 1975.

[27] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197.

[28] Benjamin Doerr. “Exponential upper bounds for the runtime of randomized search heuristics”. In: *Theoretical Computer Science* 851 (2021), pp. 24–38.

[29] Benjamin Doerr, Carola Doerr, Aneta Neumann, Frank Neumann, and Andrew Sutton. “Optimization of chance-constrained submodular functions”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 02. 2020, pp. 1460–1467.

[30] Benjamin Doerr, Edda Happ, and Christian Klein. “Crossover can provably be useful in evolutionary computation”. In: *Theoretical Computer Science* 425 (2012), pp. 17–33.

[31] Benjamin Doerr, Thomas Jansen, Dirk Sudholt, Carola Winzen, and Christine Zarges. "Mutation rate matters even when optimizing monotonic functions". In: *Evolutionary computation* 21.1 (2013), pp. 1–27.

[32] Benjamin Doerr and Andrew M Sutton. "When resampling to cope with noise, use median, not mean". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2019, pp. 242–248.

[33] Benjamin Doerr and Weijie Zheng. "Theoretical analyses of multi-objective evolutionary algorithms on multi-modal objectives: (hot-off-the-press track at GECCO 2021)". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2021, pp. 25–26.

[34] Stefan Droste. "A rigorous analysis of the compact genetic algorithm for linear functions". In: *Natural Computing* 5.3 (2006), pp. 257–283.

[35] Stefan Droste. "Analysis of the $(1 + 1)$ EA for a Noisy OneMax". In: *Genetic and Evolutionary Computation – GECCO 2004*. Ed. by Kalyanmoy Deb. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 1088–1099. ISBN: 978-3-540-24854-5.

[36] Agoston E Eiben. "Multiparent recombination in evolutionary computing". In: *Advances in evolutionary computing*. Springer, 2003, pp. 175–192.

[37] Agoston E. Eiben, P-E. Raue, and Zs. Ruttkay. "Genetic algorithms with multi-parent recombination". In: *International conference on parallel problem solving from nature*. Springer. 1994, pp. 78–87.

[38] Agoston E. Eiben and CA Schippers. "Multi-parent's niche: n-ary crossovers on NK-landscapes". In: *International conference on parallel problem solving from nature*. Springer. 1996, pp. 319–328.

[39] Hamidreza Eskandari, Christopher D Geiger, and Robert Bird. "Handling uncertainty in evolutionary multiobjective optimization: SPGA". In: *2007 IEEE Congress on Evolutionary Computation*. IEEE. 2007, pp. 4130–4137.

[40] Jesús Guillermo Falcón-Cardona and Carlos A. Coello Coello. "Indicator-based multi-objective evolutionary algorithms: A comprehensive survey". In: *ACM Computing Surveys (CSUR)* 53.2 (2020), pp. 1–35.

[41] Jonathan E. Fieldsend and Richard M. Everson. "The Rolling Tide Evolutionary Algorithm: A Multiobjective Optimizer for Noisy Optimization Problems". In: *IEEE Transactions on Evolutionary Computation* 19.1 (Feb. 2015), pp. 103–117.

[42] Jonathan E. Fieldsend and Richard M. Everson. "The rolling tide evolutionary algorithm: A multiobjective optimizer for noisy optimization problems". In: *IEEE Transactions on Evolutionary Computation* 19.1 (2014), pp. 103–117.

[43] J. Michael Fitzpatrick and John J. Grefenstette. "Genetic algorithms in noisy environments". In: *Machine Learning* 3.2 (Oct. 1988), pp. 101–120. ISSN: 1573-0565. DOI: 10.1007/BF00113893.

[44] Carlos M Fonseca, Luis Paquete, and Manuel López-Ibánez. "An improved dimension-sweep algorithm for the hypervolume indicator". In: *2006 IEEE international conference on evolutionary computation*. IEEE. 2006, pp. 1157–1163.

[45] Tobias Friedrich, Jun He, Nils Hebbinghaus, Frank Neumann, and Carsten Witt. "Approximating covering problems by randomized search heuristics using multi-objective models". In: *Evolutionary Computation* 18.4 (2010), pp. 617–633.

[46] Tobias Friedrich, Timo Kötzing, Martin S Krejca, Samadhi Nallaperuma, Frank Neumann, and Martin Schirneck. "Fast building block assembly by majority vote crossover". In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. 2016, pp. 661–668.

[47] Tobias Friedrich, Timo Kötzing, Martin S Krejca, and Andrew M Sutton. "The compact genetic algorithm is efficient under extreme Gaussian noise". In: *IEEE Transactions on Evolutionary Computation* 21.3 (2017), pp. 477–490.

[48] Tobias Friedrich, Timo Kötzing, Martin S. Krejca, and Andrew M. Sutton. "Graceful Scaling on Uniform Versus Steep-Tailed Noise". In: *Parallel Problem Solving from Nature – PPSN XIV*. Cham: Springer International Publishing, 2016, pp. 761–770. ISBN: 978-3-319-45823-6.

[49] Tobias Friedrich, Timo Kötzing, Martin S. Krejca, and Andrew M. Sutton. "The Benefit of Recombination in Noisy Evolutionary Search". In: *Algorithms and Computation*. Ed. by Khaled Elbassioni and Kazuhisa Makino. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 140–150. ISBN: 978-3-662-48971-0.

[50] Ioan Gavrea and Mircea Ivan. "On a conjecture concerning the sum of the squared Bernstein polynomials". In: *Applied Mathematics and Computation* 241 (2014), pp. 70–74.

[51] Christian Gießen and Timo Kötzing. "Robustness of Populations in Stochastic Environments". In: *Algorithmica* 75.3 (2016), pp. 462–489.

[52] Philippe Giguere and David E. Goldberg. "Population sizing for optimum sampling with genetic algorithms: A case study of the Onemax problem". In: *Genetic Programming* 98 (1998), pp. 496–503.

[53] Chi Keong Goh, Kay Chen Tan, Chun Yew Cheong, and Yew-Soon Ong. "An investigation on noise-induced features in robust evolutionary multi-objective optimization". In: *Expert Systems with Applications* 37.8 (2010), pp. 5960–5980.

[54] Chi-Keong Goh and Kay Chen Tan. *Evolutionary multi-objective optimization in uncertain environments issues and algorithms.* Springer, Berlin, Heidelberg, 2009.

[55] David E. Goldberg, Kalyanmoy Deb, and James H. Clark. "Genetic algorithms, noise, and the sizing of populations". In: *Urbana* 51 (1991), p. 61801.

[56] Heiner Gonska, Ioan Raşa, and Maria-Daniela Rusu. "Chebyshev-Gruss-type inequalities via discrete oscillations". In: *arXiv preprint arXiv:1401.7908* (2014).

[57] George Harik, Erick Cantú-Paz, David E. Goldberg, and Brad L. Miller. "The gambler's ruin problem, genetic algorithms, and the sizing of populations". In: *Evolutionary Computation* 7.3 (1999), pp. 231–253.

[58] Georges R. Harik, Fernando G. Lobo, and David E. Goldberg. "The compact genetic algorithm". In: *IEEE transactions on evolutionary computation* 3.4 (1999), pp. 287–297.

[59] Jun He and Xin Yao. "Drift analysis and average time complexity of evolutionary algorithms". In: *Artificial intelligence* 127.1 (2001), pp. 57–85.

[60] Dorit S Hochbaum. "Machine learning and data mining with combinatorial optimization algorithms". In: *Recent Advances in Optimization and Modeling of Contemporary Problems.* INFORMS, 2018, pp. 109–129.

[61] Evan J Hughes. "Evolutionary multi-objective ranking with uncertainty and noise". In: *International Conference on Evolutionary Multi-Criterion Optimization.* Springer. 2001, pp. 329–343.

[62] Thomas Jansen. *Analyzing evolutionary algorithms: The computer science perspective*. Springer Science & Business Media, 2013.

[63] Thomas Jansen and Ingo Wegener. "On the analysis of evolutionary algorithms—a proof that crossover really can help". In: *European Symposium on Algorithms*. Springer. 1999, pp. 184–193.

[64] Thomas Jansen and Ingo Wegener. "Real royal road functions—where crossover provably is essential". In: *Discrete applied mathematics* 149.1-3 (2005), pp. 111–125.

[65] Thomas Jansen and Ingo Wegener. "The analysis of evolutionary algorithms–A proof that crossover really can help". In: *Algorithmica* 34.1 (2002), pp. 47–66.

[66] Yaochu Jin and Jürgen Branke. "Evolutionary optimization in uncertain environments-a survey". In: *IEEE Transactions on Evolutionary Computation* 9.3 (2005), pp. 303–317.

[67] Rajeev Kumar and Nilanjan Banerjee. "Analysis of a multiobjective evolutionary algorithm on the 0–1 knapsack problem". In: *Theoretical Computer Science* 358.1 (2006), pp. 104–120.

[68] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. "Combining convergence and diversity in evolutionary multiobjective optimization". In: *Evolutionary computation* 10.3 (2002), pp. 263–282.

[69] Marco Laumanns, Lothar Thiele, and Eckart Zitzler. "Running time analysis of multiobjective evolutionary algorithms on pseudo-boolean functions". In: *IEEE Transactions on Evolutionary Computation* 8.2 (2004), pp. 170–182.

[70] Per Kristian Lehre and Phan Trung Hai Nguyen. "Runtime Analyses of the Population-Based Univariate Estimation of Distribution Algorithms on LeadingOnes". In: *Algorithmica* 83.10 (2021), pp. 3238–3280.

[71] Per Kristian Lehre and Phan Trung Hai Nguyen. "Runtime analysis of the univariate marginal distribution algorithm under low selective pressure and prior noise". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2019, pp. 1497–1505.

[72] Per Kristian Lehre and Xiaoyu Qin. "More precise runtime analyses of non-elitist EAs in uncertain environments". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2021, pp. 1160–1168.

[73] Johannes Lengler. "Drift analysis". In: *Theory of Evolutionary Computation*. Springer, 2020, pp. 89–131.

[74] Johannes Lengler and Jonas Meier. "Large population sizes and crossover help in dynamic environments". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2020, pp. 610–622.

[75] Johannes Lengler and Simone Riedi. "Runtime Analysis of the $(mu+1)$-EA on the Dynamic BinVal Function". In: *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*. Springer. 2021, pp. 84–99.

[76] Johannes Lengler and Ulysse Schaller. "The (1+ 1)-EA on noisy linear functions with random positive weights". In: *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2018, pp. 712–719.

[77] Efrén Mezura-Montes and Carlos A. Coello Coello. "Constraint-handling in nature-inspired numerical optimization: past, present and future". In: *Swarm and Evolutionary Computation* 1.4 (2011), pp. 173–194.

[78] Brad L. Miller. "Noise, Sampling, and Efficient Genetic Algorithms". PhD thesis. University of Illinois at Urbana-Champaign, 1997.

[79] Brad L. Miller and David E. Goldberg. “Optimal sampling for genetic algorithms”. In: *Proceedings of the artificial neural networks in engineering (ANNIE’96) conference*. Vol. 6. Citeseer. 1996, pp. 291–297.

[80] Heinz Mühlenbein. “Parallel genetic algorithms, population genetics and combinatorial optimization”. In: *Workshop on Parallel Processing: Logic, Organization, and Technology*. Springer. 1989, pp. 398–406.

[81] Heinz Mühlenbein. “The equation for response to selection and its use for prediction”. In: *Evolutionary Computation* 5.3 (1997), pp. 303–346.

[82] Heinz Mühlenbein and Gerhard Paaß. “From recombination of genes to the estimation of distributions I. Binary parameters”. In: *International conference on parallel problem solving from nature*. Springer. 1996, pp. 178–187.

[83] Frank Neumann and Andrew M Sutton. “Runtime analysis of the (1+ 1) evolutionary algorithm for the chance-constrained knapsack problem”. In: *Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*. 2019, pp. 147–153.

[84] Frank Neumann and Ingo Wegener. “Minimum spanning trees made easier via multi-objective optimization”. In: *Natural Computing* 5.3 (2006), pp. 305–319.

[85] Frank Neumann and Carsten Witt. “Combinatorial Optimization and Computational Complexity”. In: *Bioinspired Computation in Combinatorial Optimization*. Springer Berlin Heidelberg, 2010, pp. 9–19. DOI: 10.1007/978-3-642-16544-3_2.

[86] Taejin Park and Kwang Ryel Ryu. “Accumulative sampling for noisy evolutionary multi-objective optimization”. In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. 2011, pp. 793–800.

[87] Martin Pelikan and Kumara Sastry. "Initial-population Bias in the Univariate Estimation of Distribution Algorithm". In: *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*. GECCO '09. Montreal, Quebec, Canada: ACM, 2009, pp. 429–436.

[88] Martin Pelikan, Kumara Sastry, and David E. Goldberg. "Multiobjective hBOA, clustering, and scalability". In: *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. 2005, pp. 663–670.

[89] Robi Polikar. "Ensemble based systems in decision making". In: *IEEE Circuits and systems magazine* 6.3 (2006), pp. 21–45.

[90] Mojgan Pourhassan, Wanru Gao, and Frank Neumann. "Maintaining 2-approximations for the dynamic vertex cover problem using evolutionary algorithms". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 2015, pp. 903–910.

[91] Adam Prügel-Bennett, Jonathan Rowe, and Jonathan Shapiro. "Run-Time Analysis of Population-Based Evolutionary Algorithm in Noisy Environments". In: *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII*. FOGA '15. Aberystwyth, United Kingdom: ACM, 2015, pp. 69–75. ISBN: 978-1-4503-3434-1.

[92] Chao Qian, Chao Bian, Wu Jiang, and Ke Tang. "Running Time Analysis of the $(1+1)$-EA for OneMax and LeadingOnes Under Bit-Wise Noise". In: *Algorithmica* 81.2 (2019), pp. 749–795.

[93] Chao Qian, Chao Bian, Wu Jiang, and Ke Tang. "Running Time Analysis of the (1+1)-EA for OneMax and LeadingOnes Under Bit-wise Noise". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '17. Berlin, Germany: ACM, 2017, pp. 1399–1406.

[94] Chao Qian, Yang Yu, Ke Tang, Yaochu Jin, Xin Yao, and Zhi-Hua Zhou. "On the effectiveness of sampling for evolutionary optimization in noisy environments". In: *Evolutionary computation* 26.2 (2018), pp. 237–267.

[95] Chao Qian, Yang Yu, and Zhi-Hua Zhou. "An analysis on recombination in multi-objective evolutionary optimization". In: *Artificial Intelligence* 204 (2013), pp. 99–119.

[96] Chao Qian, Yang Yu, and Zhi-Hua Zhou. "Analyzing Evolutionary Optimization in Noisy Environments". In: *Evolutionary Computation* 26.1 (Mar. 2018), pp. 1–41. ISSN: 1063-6560.

[97] Pratyusha Rakshit, Amit Konar, and Swagatam Das. "Noisy evolutionary optimization algorithms – A comprehensive survey". In: *Swarm and Evolutionary Computation* 33 (2017), pp. 18–45. ISSN: 2210-6502.

[98] Magnus Rattray and Jon Shapiro. "Noisy fitness evaluation in genetic algorithms and the dynamics of learning". In: (1998).

[99] Vahid Roostapour, Aneta Neumann, and Frank Neumann. "On the performance of baseline evolutionary algorithms on the dynamic knapsack problem". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2018, pp. 158–169.

[100] Jonathan E. Rowe and Aishwaryaprajna. "The benefits and limitations of voting mechanisms in evolutionary optimisation". In: *Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*. 2019, pp. 34–42.

[101] Yasuhito Sano and Hajime Kita. "Optimization of noisy fitness functions by means of genetic algorithms using history of search with test of estimation". In: *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02Th8600)*. Vol. 1. IEEE. 2002, pp. 360–365.

[102] Yasuhito Sano, Hajime Kita, Ichikai Kamihira, and Masashi Yamaguchi. “Online optimization of an engine controller by means of a genetic algorithm using history of search”. In: *2000 26th Annual Conference of the IEEE Industrial Electronics Society. IECON 2000. 2000 IEEE International Conference on Industrial Electronics, Control and Instrumentation. 21st Century Technologies*. Vol. 4. IEEE. 2000, pp. 2929–2934.

[103] Kumara Sastry, David E. Goldberg, and Xavier Llora. “Towards Billion-bit Optimization via a Parallel Estimation of Distribution Algorithm”. In: *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. GECCO ’07. London, England: ACM, 2007, pp. 577–584.

[104] Andrey V. Savkin and Hailong Huang. “Proactive deployment of aerial drones for coverage over very uneven terrains: A version of the 3D art gallery problem”. In: *Sensors* 19.6 (2019), p. 1438.

[105] Carlos Segura, Carlos A. Coello Coello, Gara Miranda, and Coromoto León. “Using multi-objective evolutionary algorithms for single-objective constrained and unconstrained optimization”. In: *Annals of Operations Research* 240.1 (2016), pp. 217–250.

[106] Eduard Semsch, Michal Jakob, Dušan Pavlicek, and Michal Pechoucek. “Autonomous UAV surveillance in complex urban environments”. In: *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*. Vol. 2. IEEE. 2009, pp. 82–85.

[107] Vitaly Shaferman and Tal Shima. “Unmanned aerial vehicles cooperative tracking of moving ground target in urban environments”. In: *Journal of guidance, control, and dynamics* 31.5 (2008), pp. 1360–1371.

[108] Feng Shi, Martin Schirneck, Tobias Friedrich, Timo Kötzing, and Frank Neumann. “Reoptimization times of evolutionary algorithms on linear func-

tions under dynamic uniform constraints". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 1407–1414.

[109] Vui Ann Shim, Kay Chen Tan, Jun Yong Chia, and Abdullah Al Mamun. "Multi-objective optimization with estimation of distribution algorithm in a noisy environment". In: *Evolutionary computation* 21.1 (2013), pp. 149–177.

[110] Pantelimon Stanica. "Good Upper and Lower Bounds on Binomial Coefficients". In: *Journal of Inequalities in Pure and Applied Mathematics* 2.3 (2003).

[111] Dirk Sudholt. "Analysing the robustness of evolutionary algorithms to noise: refined runtime bounds and an example where noise is beneficial". In: *Algorithmica* 83.4 (2021), pp. 976–1011.

[112] Dirk Sudholt. "Crossover is provably essential for the Ising model on trees". In: *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. 2005, pp. 1161–1167.

[113] Jürgen Teich. "Pareto-front exploration with uncertain objectives". In: *International Conference on Evolutionary Multi-Criterion Optimization*. Springer. 2001, pp. 314–328.

[114] H. Vogt. "Unimodality of differences". In: *Metrika* 30.1 (1983), pp. 165–170.

[115] Yinghui Wang, Thiagalingam Kirubarajan, Ratnasingham Tharmarasa, Rahim Jassemi-Zargani, and Nathan Kashyap. "Multiperiod coverage path planning and scheduling for airborne surveillance". In: *IEEE Transactions on Aerospace and Electronic Systems* 54.5 (2018), pp. 2257–2273.

[116] Richard A. Watson and Thomas Jansen. "A building-block royal road where crossover is provably essential". In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. 2007, pp. 1452–1459.

[117] Lyndon While, Philip Hingston, Luigi Barone, and Simon Huband. “A faster algorithm for calculating hypervolume”. In: *IEEE transactions on evolutionary computation* 10.1 (2006), pp. 29–38.

[118] Darrell Whitley, Swetha Varadarajan, Rachel Hirsch, and Anirban Mukhopadhyay. “Exploration and Exploitation Without Mutation: Solving the Jump Function in $\Theta(n)$ Time”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2018, pp. 55–66.

[119] Carsten Witt. “Upper Bounds on the Runtime of the Univariate Marginal Distribution Algorithm on OneMax”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’17. Berlin, Germany: ACM, 2017, pp. 1415–1422.

[120] Zijun Wu, Michael Kolonko, and Rolf H. Möhring. “Stochastic Runtime Analysis of the Cross-Entropy Algorithm”. In: *IEEE Transactions on Evolutionary Computation* 21.4 (2017), pp. 616–628. ISSN: 1089-778X.

[121] Yue Xie, Oscar Harper, Hirad Assimi, Aneta Neumann, and Frank Neumann. “Evolutionary algorithms for the chance-constrained knapsack problem”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2019, pp. 338–346.

[122] Yue Xie, Aneta Neumann, and Frank Neumann. “Specific single-and multi-objective evolutionary algorithms for the chance-constrained knapsack problem”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 2020, pp. 271–279.

[123] Jian Zhang. “Occlusion-aware uav path planning for reconnaissance and surveillance in complex environments”. In: *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE. 2019, pp. 1435–1440.

[124] Eckart Zitzler and Lothar Thiele. “Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach”. In: *IEEE transactions on Evolutionary Computation* 3.4 (1999), pp. 257–271.

[125] Eckart Zitzler and Lothar Thiele. “Multiobjective optimization using evolutionary algorithms—a comparative case study”. In: *International conference on parallel problem solving from nature*. Springer. 1998, pp. 292–301.