Evolutionary Diversity Optimisation for Combinatorial Problems

Adel Nikfarjam
Supervisor:
Prof. Frank Neumann
Co-Supervisors:
Dr. Aneta Neumann
Dr. Jakob Bossek

A thesis submitted for the degree of
DOCTOR OF PHILOSOPHY
The University of Adelaide

November 17, 2023

Contents

List of Figures

List of Tables

University of Adelaide

Abstract

Evolutionary Diversity Optimisation for Combinatorial Problems

by Adel Nikfarjam

Diversity optimisation explores a variety of solutions for the intended problem and is rapidly growing and getting more popular within the evolutionary computation community as a result. There can be found several studies that introduce and examine evolutionary approaches to compute a diverse set of solutions for optimisation problems in the continuous domain. To the best of our knowledge, the discrete problems are yet to be studied in the context of diversity optimisation. Thus, this thesis focuses on combinatorial optimisation problems with discrete solution spaces. Here, we compute and explore such solution sets for several noticeable combinatorial problems. We aim to introduce and design evolutionary algorithms capable of computing a diverse set of solutions for the given combinatorial optimisation problem.

First, we begin with a comprehensive literature review of the recent developments and then dig deep into two prominent diverse paradigms in evolutionary computation: evolutionary diversity optimisation and quality diversity. These concepts have gained a considerable amount of attention in recent years. Quality diversity aims to achieve diversity in behavioural spaces, while evolutionary diversity optimisation sees diversity in the structural properties of solutions.

We study the evolutionary algorithms for the travelling salesperson problem, the travelling thief program, the knapsack problem, and finally, the Boolean satisfiability problem. The prospective results demonstrate the capability of the introduced algorithms to achieve diverse and high-quality solutions.

Declaration of Authorship

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Adel Nikfarjam

June 2023

Acknowledgements

I would like to express my sincere appreciation to all the individuals who have provided encouragement and assistance throughout my PhD study. Their contributions have been instrumental in enabling the completion of this study.

First and foremost, I am deeply grateful to my principal supervisor, Prof. Frank Neumann, for his constant inspiration, unwavering support, and invaluable expertise in the field. It has been an honor to learn from him and be a part of his esteemed research group.

I would also like to extend my gratitude to my co-supervisors, Dr. Aneta Neumann and Dr. Jakob Bossek, for their guidance and support throughout my research journey.

Furthermore, I would like to express my appreciation to all the co-authors of the papers, including Prof. Tobias Friedrich, Dr. Ralf Rothenberger, Viet Anh Do, and Amirhossien Moosavi. Collaborating with them has been a tremendous learning experience.

I am also grateful to my friends in Adelaide who have provided support and assistance during the past three and a half years.

Last but certainly not least, I would like to express my deepest gratitude to my beloved parents, Abbas and Touran, my caring siblings Soodeh and Ali, my kind sister-in-law Pariya, and my lovely girlfriend Nasim. Their kindness and generous support have meant the world to me.

Chapter 1

Introduction

In various real-world industries, businesses, and machines, practitioners must find a high-performing setting in their systems to perform desirably. Optimisation is the procedure of finding the best or a high-quality solution (setting) out of all possible solutions with respect to an objective function. In many systems, an optimisation problem can be found that needs to be addressed in pursuit of better performance. Therefore, we can observe optimisation problems in various subjects, from computer science and engineering to health and educational systems.

Various algorithms have been proposed to solve optimisation problems, depending on the problems' characteristics, the size of the problem, and the time budget we have for solving it. These algorithms range from exact algorithms to approximation methods, heuristics and metaheuristics. Exact algorithms are methods to solve an optimisation problem to optimality and provide the decision-makers with the proven global optimum, referred to as the best possible solution in the entire solution space. However, many algorithms in this class are not time-efficient and cannot solve many real-world problems in a practical time window.

Heuristics and metaheuristics, on the other hand, produce high-quality solutions in a reasonable time window, but they cannot guarantee the optimality of solutions. This class of algorithms often aim for a high-quality solution, although these algorithms may find a local or global optimum. A local optimum is referred to as the best solution compared to their neighbouring solutions. Examples of these algorithms include simulated annealing [54], ant colony optimisation [31], evolutionary algorithms [64], among others. This thesis focuses on bio-inspired metaheuristics, particularly evolutionary algorithms (EAs). EAs are iterative population-based algorithms where a population of solutions compete for reproduction and survival. In each iteration, several offspring solutions are generated; then, old and young solutions compete to survive for the next generation.

Classical optimisation algorithms often aim for a single global or local optimal solution [14, 43]. A few concepts have evolved around finding multiple diverse solutions for an optimisation problem in the literature on evolutionary computation (EC), such as multimodal optimisation, quality diversity, and evolutionary diversity optimisation. However, these concepts are considerably less studied than the conventional approaches, especially evolutionary diversity optimisation (EDO) and quality diversity (QD). QD sees diversity in computing best-performing solutions with different, diverse behaviours, while EDO aims to diversify the structural properties of solutions, such as edges included in a solution for the travelling salesperson problem (TSP) or items included in a packing list, subject to the quality of solutions.

Several studies in the literature can be found investigating QD approaches in the context of optimisation problems in the continuous domain, such as robotics and games [36, 96]. Also, several studies can be found in the literature on EDO that focus on diversity in a continuous feature space such as images or benchmarking instances [2, 77, 78]. However, investigating the performance of QD-based methods and EDO approaches in combinatorial optimisation problems is neglected. Combinatorial optimisation problems are problems with a finite set of possible solutions where solution space is discrete. This lack of studies motivates us to conduct this research work. Within this thesis, we aim to design and introduce EAs based on QD and EDO frameworks to compute a diverse set of solutions for combinatorial optimisation problems.

Finding a diverse set of high-quality solutions has several clear advantages. Here, we mention a few critical merits to highlight the importance of diversity in solutions. Firstly, computing a diverse set of high-quality solutions provides researchers and practitioners with invaluable information about the solution space. For instance, how high-quality solutions look like in the problem under consideration or which components of high-quality solutions are irreplaceable. Secondly, such sets of solutions boost the robustness of algorithms against imperfect modelling and minor changes in the problem. In TSP, for instance, if an edge becomes unavailable for some reason, a solution that excludes that particular edge becomes beneficial; having such a solution helps us to avoid re-computation. Finally, it provides the decision-makers with alternative options instead of imposing them with a single solution. Having several solutions at hand enables decision-makers to discuss and consider the preferences of different stakeholders who may have conflicting interests in some cases and select a solution accordingly.

1.1 Our Contribution

This thesis consists of nine following chapters. Chapter 1 and 2 are dedicated to an introduction and defining preliminaries, respectively. The rest seven chapters come as follows: Chapter 3 employ a population diversity measure, called the high-order entropy measure, in an evolutionary algorithm to compute a diverse set of high-quality solutions for the TSP. In contrast to previous studies, our approach allows diversifying segments of tours containing several edges based on the entropy measure. We examine the resulting evolutionary diversity optimisation approach precisely in terms of the final set of solutions and theoretical properties. Experimental results show significant improvements compared to a recently proposed edge-based diversity optimisation approach when working with a large population of solutions or long segments.

Moreover, most studies in the literature on EDO used a specific EA where only one offspring solution is created in each generation. The rationale behind this is rooted in the complex nature of diversity calculation. Increasing the number of offspring solutions raises the computational cost of survival selection substantially. In this chapter, we propose three survival selections to overcome this issue. The results show that the EA-based survival selection outperforms the commonly used EA.

Evolutionary algorithms based on edge assembly crossover (EAX) [72] constitute some of the best-performing incomplete solvers for the well-known TSP. Currently, there are only a few approaches for computing a diverse solution set for the TSP. Furthermore, almost all of them assume that the optimal solution is known. Chapter 4 introduces EDO-based approaches for the TSP that find a diverse set of tours when the optimal tour may be unknown. We show how to adopt EAX not only to find a high-quality solution but also to maximise the diversity of the population. The resulting EAX-based EDO approach, termed EAX-EDO, is capable of obtaining diverse, high-quality solutions when the optimal solution for the TSP is known or unknown. Comparison to existing approaches shows that EAX-EDO clearly outperforms them.

In real-world optimisation, it is common to face several sub-problems interacting and forming the main problem. There is an inter-dependency between the sub-problems, making it impossible to solve such a problem by focusing on only one component. The travelling thief problem (TTP) belongs to this category and is formed by the integration of the TSP and the knapsack (KP). In Chapter 5, we investigate the inter-dependency of the TSP and the KP by means of QD approaches. QD algorithms provide a powerful tool not only to obtain high-quality solutions but also to illustrate the distribution of high-performing solutions in a behavioural space.

We introduce a QD-based evolutionary algorithm using the well-known TSP and KP search operators, taking the TSP and KP score as the behavioural descriptor (BD). Afterwards, we conduct comprehensive experimental studies that show the efficiency of the QD approach applied to the TTP. First, we provide insights regarding high-quality TTP solutions in the TSP/KP behavioural space. Then, we show that better solutions for the TTP can be obtained by using our QD approach, and it can improve the best-known solutions for a number of TTP instances used for benchmarking in the literature.

Chapter 6 investigates a prominent multi-component optimisation problem, the

TTP, in the context of EDO for the first time. We introduce a bi-level evolutionary algorithm to maximise the structural diversity among the TTP solutions. Moreover, we examine the inter-dependency between the problem's components and investigate its impact on structural diversity solutions. We empirically determine the best method to maximise diversity and conduct a comprehensive experimental investigation to examine the introduced algorithm. We also compare the results to the QD-based framework introduced in Chapter 5. Our experimental results show a significant improvement in the QD approach in terms of structural diversity for most TTP benchmark instances.

The chapter also presents a co-evolutionary algorithm to simultaneously explore the two spaces for the multi-component travelling thief problem. Co-evolutionary algorithms are methods that let several populations evolve in parallel and interact with each other. The results show the capability of the co-evolutionary algorithm to achieve significantly higher diversity compared to the EDO-based algorithm introduced earlier.

As mentioned, QD algorithms have been shown to be very successful when dealing with problems in areas such as robotics and games. They aim to maximise the quality of solutions for different regions of the so-called behavioural space of the underlying problem. Chapter 7 applies the QD paradigm to simulate dynamic programming behaviours on the KP and provides a first runtime analysis of QD algorithms. We show that they are able to compute an optimal solution within expected pseudo-polynomial time and reveal parameter settings that lead to a fully polynomial randomised approximation scheme (FPRAS). Our experimental investigations evaluate the different approaches on classical benchmark sets in terms of solutions constructed in the behavioural space as well as the runtime needed to obtain an optimal solution.

Chapter 8 studies the Boolean satisfiability problem (SAT) in the context of EDO. The SAT is one of the most important problems in computer science due to its applications. The SAT also differs from the other problems studied in the literature of EDO and previous chapters, such as the KP and the TSP. SAT is heavily constrained, and random operators are incapable of generating feasible SAT solutions. Therefore, we introduced an EA-based approach with the following features: 1) it iteratively adds and removes constraints (clauses) to the problem to forbid solutions or to fix variables; 2) it incorporates powerful solvers in the literature on SAT, such as minisat into the EA to construct a diverse set of SAT solutions. The EA-based method explicitly maximises diversity among a set of SAT solutions. Experimental investigations show the introduced algorithm's capability to maximise diversity among the SAT solutions. Finally, Chapter 9 is dedicated to some concluding remarks and suggestions for future studies and the extension of this research.

1.2 Underlying Publications

This subsection is dedicated to providing the list of the papers published from this PhD research work:

- Chapter 3: Adel Nikfarjam, Jakob Bossek, Aneta Neumann, and Frank Neumann. "Entropy-based evolutionary diversity optimisation for the traveling salesperson problem". In: GECCO. ACM, 2021, pp. 600–608.

- Chapter 4: Adel Nikfarjam, Jakob Bossek, Aneta Neumann, and Frank Neumann. "Computing diverse sets of high quality TSP tours by EAX-based evolutionary diversity optimisation". In: FOGA. ACM, 2021, 9:1–9:11.

- Chapter 5: Adel Nikfarjam, Aneta Neumann, and Frank Neumann. "On the use of quality diversity algorithms for the traveling thief problem". In: GECCO. ACM, 2022, pp. 260–268.

- Chapter 6: Adel Nikfarjam, Aneta Neumann, and Frank Neumann. "Evolutionary diversity optimisation for the traveling thief problem". In: GECCO. ACM, 2022, pp. 749–756. Adel Nikfarjam, Aneta Neumann, Jakob Bossek, and Frank Neumann. "Co-evolutionary Diversity Optimisation for the Traveling Thief Problem". In: PPSN(1). Vol. 13398. Lecture Notes in Computer Science. Springer, 2022, pp. 237–249.

- Chapter 7: Adel Nikfarjam, Anh Viet Do, and Frank Neumann. "Analysis of Quality Diver-sity Algorithms for the Knapsack Problem". In: PPSN (2). Vol. 13399. Lecture Notes in Computer Science. Springer, 2022, pp. 413–427.

- Chapter 8: Adel Nikfarjam, Ralf Rothenberger, Frank Neumann, and Tobias Friedrich."Evolutionary Diversity Optimisation in Constructing Satisfying Assignments". In: GECCO. ACM, 2023.

Chapter 2

Basics

This chapter is dedicated to defining the preliminaries and reviewing the background of diversity in evolutionary computation literature, with an orientation towards evolutionary diversity optimisation (EDO) and quality diversity (QD). The chapter is structured as follows: Section 2.1 looks into the notions of optimisation, heuristics, and meta-heuristic algorithms. Section 2.2 introduces EAs, while Section 2.3 reviews the literature on diversity in EC. Section 2.4 defines the combinatorial problems that are studied in this research. Finally, we finish with some concluding remarks.

2.1 Optimisation and Heuristics

Optimisation can be defined as the process of computing one of the best possible solutions from the set of all feasible solutions to the problem at hand. Optimisation problems are expanded in numerous domains, such as computer science, operations research, logistics and engineering. Optimisation problems usually consist of an objective function and a set of constraints that impose some limitations or conditions on the problem. A typical optimisation problem can be formulated as follows:

$$\begin{aligned}
&\text{Minimise} \quad Z(x)\\
&\text{Subject to:}\\
&\omega(x) \quad \forall \omega \in \Omega.
\end{aligned}$$

Here, x denotes a solution to the problem and $Z(x)$ is a function quantifying the objective. Also, Ω denotes the set of constraints of the problem. A typical constraint usually is an inequality or an equality function. The formula is a minimisation problem such as minimising cost, waste, or waiting times. Note that minimisation is no restriction as minimising Z is equivalent to maximising $-Z$.

In practice, it is often challenging to compute one of the best possible solutions, called optimal solutions, for a real-world problem within a reasonable time window.

Thus, researchers and practitioners often turn to heuristic and metaheuristics solution approaches instead of exact algorithms that result in proven optimal solutions. On the contrary, metaheuristics algorithms cannot guarantee the optimality of solutions. Nevertheless, these types of algorithms are able to find high-quality solutions in a reasonable amount of time. Metaheuristics are referred to as a strategy of iteratively creating and improving solutions to find near-optimal solutions [91]. Heuristics and metaheuristics such as EAs, local search, simulated annealing, and ant colony optimisation have been shown to be suitable for optimisation problems, and in particular for combinatorial optimisation problems [1].

2.1.1 Local Search

Local search algorithms are well-known metaheuristics often employed for solving complex optimisation problems [1]. Unlike exact algorithms, local search methods are designed to explore neighbourhoods and find a local optimum rather than aiming for the global optima. Although these algorithms cannot result in proven global optima, they have received increasing attention due to their performance in solving computationally complex problems. Local search works on an initial solution and iteratively improves it by searching the neighbouring solutions and replacing the current solution with another one, usually with a solution with better or the same quality.

Hill climbing is a method used in solving optimisation problems that belong to the class of local search. Hill climbing starts with an initial algorithm and iteratively generates new solutions by making small changes to the solution. If the new solution has a better objective value, it is replaced with the old solution. These steps continue until there is no further improvement. Hill climbing algorithms are capable of solving convex problems to optimality. For other problems, they can only compute a local optimum. The greedy algorithms belong to this class of algorithms, where the move with the most significant improvement of the objective value is selected in each iteration.

Although hill climbing is a powerful tool in finding local optima, they are incapable of escaping one. Getting stuck in local optima can be considered the most critical drawback of hill climbing. A plausible approach to overcome this limitation is the notion of accepting downhill moves. Several methods in the class of local search utilise such a notion. One of these algorithms is simulated annealing [54].

Simulated annealing is a local search algorithm that goes beyond finding local optima and is considerably used to approximate the global optimum for the optimisation problem at hand. Simulated annealing is inspired by metallurgy, a method used to change the physical properties of metals by heating them and then cooling them down gradually. Similarly, simulated annealing accepts downhill moves mainly at the early stages. Downhill moves are referred to as accepting solutions with a worse objective value.

Algorithm 1 Simulated Annealing

```
Require: Initial temperature $t$, cooling schedule $CS(t)$, stopping temperature $T$.
 1: Generate an initial solution $x$.
 2: $x^* \leftarrow x$.
 3: while $t \geq T$ do
 4:     Generate a random neighbourhood solution $x'$.
 5:     $\delta \leftarrow f(x') - f(x)$.
 6:     if $\delta \leq 0$ then
 7:         $x \leftarrow x'$.
 8:         if $f(x) \leq f(x^*)$ then
 9:             $x^* \leftarrow x$.
10:     else
11:         Generate a number $v$ uniformly at random between 0 and 1.
12:         if $v < \exp\left(\frac{-\delta}{t}\right)$ then
13:             $x \leftarrow x'$.
14:     $t \leftarrow CS(t)$.
```

The motivation behind simulated annealing and accepting downhill moves is to escape the local optima and explore a broader range of the solution space. Simulated annealing has been successfully employed in several applications [55]. In simulated annealing, there is a parameter named temperature that controls the rate of accepting downhill moves. Temperature is typically high at the early stage of the search, and then it decreases based on a cooling down schedule so as the probability of accepting worse solutions, such that the algorithm can converge to high-quality solutions. The downhill moves and the cooling down schedule are necessary for escaping local optima and the convergence of the algorithm. Algorithm 1 represents a typical simulated annealing algorithm.

The algorithm should be passed an initial solution x, an initial temperature t, a cooling schedule function $CS(t)$, and a stopping temperature (T). The cooling function $CS(t)$ reduces t, such that the algorithm decreases the acceptance rate of worse solutions. First, x^*, which holds on the best solution found so far, is set to x. Next, a random neighbourhood solution x' is generated, and the difference between the $f(x')$ and $f(x)$ is stored in δ. If $\delta \leq 0$, x is replaced by x', and x^* is also replaced with x' in case x' is the best solution so far. Otherwise, a random number v between 0 and 1 is generated. If $v < \exp\left(\frac{-\delta}{t}\right)$, x is set to x'. Afterwards, the temperature t cools down according to $CS(t)$. These steps are repeated until $t < T$.

Iterated local search (ILS) [60] and tabu search [42] are other local search algorithms introduced to avoid getting stuck in local optima. Tabu search benefits form a list that stores the recently visited solutions to avoid the algorithm revisiting them for the sake of saving time and computational affords, as well as encouraging accepting new solutions and exploring new neighbourhoods. However, it is possible to override the tabu status of a solution according to some criteria, such as the possibility of

Algorithm 2 Iterated Local Search

```
Require: Initial solution $x$.
1: Perform a local search on $x$.
2: while a termination criterion is not met do
3:     Perform perturbation to solution $x$ and store it in $x'$
4:     Perform a local search on $x'$.
5:     if $Z(x') \leq Z(x)$ then
6:         Replace $x$ with $x'$
```

a solution leading to a more promising area. Moreover, tabu search accepts downhill moves if there cannot be found a solution with a higher objective value in the neighbourhood.

ILS is another metaheuristics algorithm within the domain of local search. ILS can successfully overcome the conventional local search limitation, meaning escaping local optima. ILS iteratively switches between the local search phase and the perturbation phase. The local search phase is similar to hill climbing, where a solution is iteratively improved until no further improvements can be found. In the perturbation phase, changes are made to a local optimum solution through rules that ensure leading the solution to a different neighbourhood of solution space. The perturbations assist the algorithm in switching from an explored neighbourhood to a new region. Algorithm 2 represents a typical ILS algorithm. The algorithm starts with an initial solution x. First, the solution goes through a round local search phase, which means a local search algorithm is applied to the solution and improves iteratively until no further improvement can be made. Then, the solution goes through a perturbation phase and generates a new solution x'. Another round of local search applies to x' until a local optimum is found. If $Z(x') \leq Z(x)$, solution x' is replaced with x. These steps continue until a termination criterion is met.

2.1.2 Swarm Intelligence Algorithms

Swarm intelligence algorithm is another class of bio-inspired population-based metaheuristic approaches that have been widely applied to optimisation problems. The term of swarm intelligence first has been introduced by Beni and Wang [7]. Swarm intelligence algorithms mimic collective behaviours of biological systems found in nature to solve optimisation problems. Several well-known metaheuristics in the literature are categorised in this class, including stochastic diffusion search [75], artificial swarm intelligence [99], ant colony optimisation [31], and particle swarm optimisation [53]. Here, we introduce ant colony optimisation (ACO) [31], and particle swarm optimisation (PSO) [53].

ACO draws inspiration from the collective behaviours of ants in nature. Ants face the problem of finding food in their everyday lives. To address the problem and find the shortest route possible from their nest to the food sources, ants incorporate

Algorithm 3 Ant Colony optimisation

```
1: Initialise pheromone trails and ant solutions $P = \{x_1, \cdots, x_\mu\}$.
2: $x^* \leftarrow x_i$ where $\arg\min_{x_i \in P} Z(x_i)$.
3: while a termination criterion is not met do
4:    for each ant $(x_i \in P)$ do
5:       while an ant solution is not complete do
6:          Choose one neighbour at each step probabilistically.
7:       if $Z(x_i) \leq Z(x^*)$ then
8:          $x^* \leftarrow x_i$.
9:    Evaporate the pheromone trails.
10:   Deposit the pheromone trails.
```

an interesting system to communicate with each other and direct others towards food. Ants leave a substance called pheromone on the way back to their nest carrying foods. The pheromone attracts the other ants and guides them towards the food source. As the number of ants coming back from a food source increases, more pheromone is left for the other ants.

ACO incorporate a similar concept into finding high-quality solutions for the optimisation problems that can be mapped to finding the shortest paths through a graph. Artificial ants, so-called agents, choose their next move randomly based on artificial pheromones. After the solution is constructed, the ants leave the pheromone on the path they took based on the quality of the solution. Through the collective efforts of artificial ant, the density of high-quality solutions' pheromone increase and the algorithm converges. Similar to nature, pheromone evaporates in ACO over time. The evaporation of the pheromone aids the algorithm in escaping local optima and avoiding premature convergence. Algorithm 3 outlines a typical ACO algorithm.

The algorithm starts with an initial population of solutions (ants) and pheromone trails. First, the best solution in the population is stored in x^*. Then, for each ant within the population ($x_i \in P$), neighbours are probabilistically selected one by one based on pheromone trails and heuristic information. If $Z(x_i) \leq Z(x^*)$, x^* is replaced with x_i. Having repeated these steps for all ants in the population, we evaporate, then deposit the pheromone trails on all edges. These steps continue until a termination criterion is met. We refer the interested reader to López-Ibáñez, Stützle, and Dorigo [59] for an overview of ACO.

Particle swarm optimisation (PSO) [53] is another well-known population-based metaheuristics in the category of swarm intelligence. PSO is inspired by the behaviour of bird flocks, humans, or fish schools. Birds and fish move in groups to find food and avoid predators. Birds, similarly to humans, try to learn from their experiences and from the ones within the group who perform well.

PSO employs a similar concept in solving problems. Individuals are called particles in PSO, representing a solution to the problem. Particles explore the solution space and use the information obtained from their previous positions and shared by other particles in the population, the so-called swarm. Particles tune their position based on their personal (local best) position and the best position found so far within the swarm (global best). Paricles are also allocated a velocity changing over time to adjust their position. As the algorithm continues, the swarm tends to converge towards high-quality solutions. A drawback of PSO is that the algorithm tends to converge so fast and get stuck in local optima. We refer interested readers to Bonyadi and Michalewicz [8] for a comprehensive review of PSO.

EAs are another well-known bio-inspired concept in the class of Population-based metaheuristics. Since this research focuses on EC, we formally introduce EAs in a separate section.

2.2 Evolutionary Algorithms

EAs are well-known population-based bio-inspired metaheuristics that are applied to many optimisation problems in various fields ranging from art and logistics to engineering and computer science [63]. As its name indicates, EAs draw inspiration from evolution mechanisms in nature, such as the selection of the fittest and reproduction [64].

EAs use a fitness function, which is often the objective function to quantify the quality of solutions and measure their fitness to be selected to seed the next generation or survive within the population. EAs also utilise bio-inspired operators to generate a new generation, such as mutation and crossover. The underlying principle is that the environment's pressure causes a natural selection within the population, leading to a new generation enhancing the population's fitness [33]. This process continues until a termination criterion is met.

2.2.1 Components of Evolutionary Algorithms

EAs start with an initial set of solutions (individuals) that resembles a population of species, such as humans or animals in nature. These individuals compete with each other to take part in reproduction and the creation of offspring. Like nature, where stronger animals have a higher chance to mate, fitter individuals often stand a higher chance to be selected and pass their genes. For reproduction, EAs use some variation operators inspired by nature to combine, mix, and mutate the parent individuals to generate new ones. Then, all individuals, including parents and offspring, fight for survival. Once again, the fitter individuals, the higher their chance for survival.

Let us go back to the optimisation formula above; EAs employ a fitness function $Z(x)$ to measure the quality of x to determine its probability of being selected to serve as a parent or to survive to the next generation. Algorithm 4 represents the sketch of a typical EA. The algorithm starts with an initial population. The diversity

Algorithm 4 Evolutionary Algorithm

```
1: Initialise a population of solutions $P = \{x_1, \cdots, x_\mu\}$.
2: $x^* \leftarrow x_i$ where $\arg\min_{x_i \in P} Z(x_i)$.
3: while a termination criterion is not met do
4:     Select parents.
5:     Apply variation operators (crossover and mutation) to generate offspring.
6:     Select the next generation.
7:     if $\arg\min_{x_i \in P} Z(x_i) \leq Z(x^*)$ then
8:         $x^* \leftarrow x_i$.
```

of the initial population plays a crucial role in covering a broad range of the solution space. Poorly diverse initial solutions can cause premature convergence, while a highly diverse population may jump between the different regions through recombination and mutation. Next, through a selection process, a number of individuals serve as parents and take part in generating offspring individuals using variation operators. Then, the next generation is selected from a pool of old individuals and offspring. These steps continue until a termination criterion is met.

Representation

EAs are powerful methods to address optimisation problems in a broad range of domains. However, they can only generate feasible, high-quality results if the problem is appropriately modelled. The first challenge that algorithm engineers encounter is a plausible design for solution representation. Many solution representations are commonly used in EAs, such as bit string, real value vector, and permutation. The algorithm designers should take into account the characteristics and features of the problem at hand and design or select a solution representation that maps all conditions of the problem onto the solutions.

A poor choice can result in imperfect modelling and eliminating regions of feasible solution space that can obtain high-performing solutions. This is why good design can not only boost the performance of the EAs, but also variation operators can benefit from it [101]. For instance, one may use binary variables to represent TSP solutions. However, the success rate of random variation operators decreases since they can quickly create an infeasible solution. This problem can be solved by using permutation representation and their specific operators such that the infeasible solution space is eliminated without reducing the feasible space. Thus, the choice of variation operators should also be considered when designing a solution representation. Another aspect in the design of solution representation we can take into consideration is their memory computation which can considerably affect the time efficiency of algorithms.

Variation Operators

Variation operators are arguably one of the most important components of an effective EA. These operators are responsible for the generation of new individuals. They operate on the parents and create offspring to give the population new blood for exploring the solution space. Therefore, it is of great importance to design operators to make a balance between exploration and exploitation. The balance can only be achieved by operators fitting to the solution representing. There are two classes of variation operators, crossover and mutation.

Crossovers are inspired by mating animals in nature. Like in nature, crossovers combine and mix the genes and components of parents to create offspring. Most crossover in the literature takes two parents for operation although some variants require more than two parents. As mentioned, variation operators should fit the representation, and crossovers are no exceptions.

Some well-known crossovers in the domain of bit strings include single-point, double-point [18, 34], and uniform crossovers [109]. For instance, consider solution $x = (x_1, \cdots, x_n)$ and $y = (y_1, \cdots, y_n)$; the single-point crossover selects a point $i \in \{1 \cdots n-1\}$ uniformly at random, cut both parent from the point i, and connect them alternatively. Then, we have two offspring individuals $o_x = (x_1 \cdots, x_i, y_{i+1}, \cdots, y_n)$ and $o_y = (y_1 \cdots, y_i, x_{i+1}, \cdots, x_n)$. The double-point crossover cuts the parents from two points, while uniform crossover selects bit $j \forall j \in \{1 \cdots n\}$ from $\{x_j, y_j\}$ uniformly at random.

The above crossovers are mainly used on bit-string solutions. The use of them on permutation mainly results in infeasible solutions since the chance of generation of a permutation with those crossovers is small. Some specific considerations are required to design crossovers for permutations so as to enforce the feasibility of solutions. Thus, this class of crossovers often tend to be more complicated. Some popular permutation crossovers include cycle crossover [90], order crossover [18], and edge crossover [118]. Here, we describe the steps involved in the implementation of order crossover:

- Select two individuals p_1 and p_2 from the population to serve as the parents.
- Randomly select segments from P_1.
- Transfer the selected segments to the offspring o in the same position as p_1.
- Identify the missing components in o.
- Transfer the missing components to the empty positions in o in the order they appear in $p2$.

Unlike crossover, mutations work on a singular parent and make small changes to generate offspring. Sometimes, a mutation crossover is applied to random offspring to introduce the elements that no individuals in the population possess. This process is similar to mutated genes in newborn babies in mammals. It is also possible to use a mutation operator directly to a parent. This process can be compared to parthenogenesis in nature. Like parthenogenesis, the offspring is similar to the parent. The standard bitflip mutation is a well-known mutation operator in the literature, where each bit of the parent flips with the probability of $\frac{1}{n}$.

Selection Methods

Last but certainly not least, selection methods are a crucial part of EAs since EAs are developed around the notion that nature selects fitter individuals to flourish in their environments. Here, EAs use a function to measure the fitness of individuals and select individuals for reproduction and survival through some mechanisms based on their fitness. The objective function $Z(x)$ often serves as the fitness function. However, the objective function may be integrated with other features, such as a diversity measure and penalty functions and form the fitness function in some cases.

Selection occurs in two places in EAs, parent selection and survival selection. Parent selection is the procedure that selects individuals to go through reproduction. Parent selection may consider the fitness of function owing to a higher chance of fitter individuals in the construction of high-quality offspring. This bias increases the chance of passing good genes of high-quality solutions to the next generation. Nevertheless, parent selection must be a probabilistic procedure and give a chance to the individuals with worse fitness to serve as parents. The motivation behind it is rooted in the fact that sometimes worse quality solutions possess unique features; combining those features with other solutions may produce decent results.

The second selection phase is called survival selection, a mechanism to select the next generation from a pool of old and offspring individuals. Survival selection is a sensitive procedure and can substantially affect the performance of EAs. If the selection is highly biased towards high-quality individuals, the algorithm may converge too fast and get stuck in a local optimum. On the other hand, if an EA has an unbiased survival selection, it never converges. Thus, the selection pressure controls the balance of exploration and exploitation. Both deterministic and probabilistic survival selection mechanisms can be found in the literature.

Proportional selection, the so-called roulette wheel, can be considered a well-known mechanism in EAs. Proportional selection determines the probability of selecting a solution x based on its fitness function $Z(x)$. The chance of selecting individual x_i from the population P can be calculated from the following formula:

$$pr(x_i) = \frac{Z(x_i)}{\sum_{j=1}^{\mu} Z(x_j)}$$

Where μ is the cardinality of set P. There is a clear bias towards the high-quality of solutions in the fitness proportional selection, which makes it a better choice for parent selection. Nevertheless, the selection can also be incorporated into the survival selection. Also, it should be noted that there are some limitations with the formula; for instance, the fitness values should always be positive.

k-tournament selection is another commonly used selection in the literature of EC. The mechanism selects k individuals uniformly at random; then, the solution with the highest fitness survives to the next generation and is removed from the pool. These steps are repeated until the following generation population forms. Here, k controls the selection pressure. As k increases, so does the selection pressure and a decrease of k raises the selection's randomness.

Termination criteria may be considered another component of EAs. There can be several termination criteria, such as the number of generations, fitness evaluations, and achieving a desirable objective value. In many cases, termination criteria are set considering the computational budget.

2.3 Diversity in Evolutionary Computation

Traditionally, among researchers in the EC community, diversity is seen as a means to explore local optima, also called niches in the fitness landscape. The process of finding a set of local optima in a single try is referred to as multimodal optimisation. EAs have been shown to be very successful in addressing multimodal optimisation problems if niching methods are incorporated. Niching methods are strategies to preserve diversity among the individuals of a population. Here, diversity is mainly seen as solutions with different fitness values.

Several niching methods can be found in the literature on multimodal optimisation. The classic niching methods include crowding techniques [52] and fitness sharing [49], which were introduced in the early 80s. Fitness sharing methods divide the population into several subpopulations according to the resemblance of solutions. The method draws inspiration from sharing notions in nature, where individuals in an environment have access to limited resources and need to save and share it among all living there.

Crowding techniques randomly draw a number of individuals from the population and remove the one with the most similarity to the others. Crowding techniques were first introduced as a means to preserve the diversity of the population to prevent premature convergence. Then, they have been widely studied in the context of multimodal optimisation.

Some other niching methods include restricted tournament selection [46], spatial [92] and density-based techniques [125]. Interested readers are referred to [103, 107] for well-established niching techniques. Also, Li et al. [57] provides a comprehensive review of recent developments in niching methods. Niching and multimodal optimisation have been extensively studied in the literature. In this research, we focus on two other paradigms that have recently evolved in the EC community in the context of diversity, QD and EDO, which have drawn less attention than niching.

2.3.1 Quality Diversity

QD [94, 95] is a recent but well-established paradigm in the field of EC that has gained increasing attention in the literature. QD sees diversity as a means to extensively explore the niches present in a pre-defined behavioural space of the optimisation problem under consideration. QD algorithms aim to find the best-performing solutions with distinctive behaviours.

Behavioural descriptors (BDs) are a critical component of QD algorithms. BDs can be a vector representing the behaviours of a solution or, in other words, the solution's position in the pre-defined behavioural space. In QD algorithms, solutions with similar BDs only compete with each other to survive for the next generation. This selection procedure ensures that solutions with unique behaviours are not replaced by each other and increases the behavioural diversity of solutions.

MAP-Elite algorithm is the best-known method in the literature of QD. Here, the behavioural space is discretised into a grid. A solution can be placed into the cell it belongs to, which is determined based on its BDs. If the cell is already taken by another individual, the offspring and old individuals compete for survival, and the solution with a higher objective value occupies the cell. QD algorithms, particularly MAP-Elite, are highly efficient in illustrating the distribution of high-quality solutions to an optimisation problem over a behavioural or feature space. Moreover, this class of EAs has also been shown surprisingly effective in finding solutions with decent objective values in many fields such as robotics and gaming [36, 96] owing to its unique diversity mechanism.

QD has emerged from the concept of novelty search, where algorithms aim to find new behaviours without considering fitness [56]. Cully and Mouret [21] introduced a mechanism to only keep the best-performing solutions while seeking new behaviours. Concurrently, Clune, Mouret, and Lipson [17] proposed a simple algorithm to plot the distribution of high-quality solutions over a feature/behavioural space. Interestingly, the proposed algorithm, named MAP-Elites, efficiently evolves behavioural repertoires. Pugh, Soros, and Stanley [94] and Pugh et al. [95] formulated the concept of computing a diverse set of high-quality solutions differing in features or behaviours and named it QD.

The paradigm has been widely applied to the areas of robotics [3, 96, 126] and games [35, 36, 108] as well as other continuous problems such as urban design [40]. In addition, the use of QD algorithms to evolve a diverse set of instances for the TSP has been studied in [13]. We refer interested readers to the review paper of Chatzilygeroudis et al. [16]. To the best of our knowledge, QD algorithms have not previously been used to solve and study a combinatorial optimisation problem until conducting this research. Therefore, this gap motivates us to provide the first study on this subject.

2.3.2 Evolutionary Diversity Optimisation

Unlike niching and QD, EDO explicitly maximises the structural diversity of the population subject to the quality of solutions. EDO employs a measure to quantify the diversity among individuals and use it as the fitness function. Also, EDO treats the objective value or the quality of solutions as constraints.

Consider the minimisation problem in Section 2.1 with an objective function $Z(x)$ and a set of constraints $\omega \in \Omega$. The goal of EDO is to maximise the diversity of the set of solutions $P = \{x_1, \dots, x_\mu\}$. For this purpose, it utilises a diversity measure function $D(P)$ as the fitness function while solution x must comply with the quality constraints $Z(x)|x \in P$.

Formally, the EDO problem can be defined as follows:

$$\begin{aligned}
&\text{Maximise} \quad D(P) \\
&\text{Subject to:} \\
&Z(x_j) \leq (1+\alpha) \cdot \text{OPT} \quad \forall x_j \in P \\
&\omega(x_j) \quad \forall \omega \in \Omega, \forall x_j \in P.
\end{aligned}$$

Here, OPT represents the optimal value of the problem at hand, and α is a factor that controls the quality pressure of the solutions. The objective is to maximise the diversity among the solutions subject to the objective values of the solutions in P $(Z(x) \leq (1+\alpha) \cdot \text{OPT}|x \in P)$. Obviously, the other constraints of the problem must be satisfied by each solution in P.

EDO makes sure we get the most diverse set of solutions possible in the acceptable range of objective values through this unique formulation. However, a drawback of this formulation is that it most likely results in solutions with objective values close to $(1+\alpha)$, and better quality solutions are usually lost.

EDO was first introduced by Ulrich and Thiele [112]. In recent years, many of the studies in the context of EDO focused on generating diverse sets of images and benchmark instances for the TSP [2, 41]. They generated sets of similar images differing in aesthetics and TSP instances with regard to the characteristics of the problem that make an instance easy or difficult to solve for different TSP solvers. In terms of different diversity measures, studies evolving diverse sets of TSP instances and sets of images by using the star-discrepancy measure [77] and indicators from multi-objective optimisation [78] have been carried out recently. Specific mutation operators to achieve high diversity in TSP instances without using diversity preservation mechanisms explicitly have been introduced in [11].

To the best of our knowledge, only Do et al. [24] investigated EDO to compute diverse sets of tours for the TSP before the conduction of the research of this thesis. Do et al. [24] introduced two distance-based diversity measures and studied simple EAs with k-OPT neighborhood search operators exclusively. The authors introduced two different diversity measures, edge diversity (ED) and pairwise distance (PD), based on pairwise edge overlap. They embedded k-OPT mutation operators with different values of k in an EA introduced to solve the EDO problem, and empirically studied the mutations' impact on the EA performance. They also assumed that the optimal values of the TSP instances are known. The lack of comprehensive studies in EDO motivates us to concentrate on this subject and study it in the context of combinatorial optimisation problems.

During the conduction of our research, other studies investigated EDO in generating diverse solutions. Do et al. [25] theoretically investigated diversity optimisation in permutation problems such as the TSP and quadratic assignment problem. Also, Bossek and Neumann [12] investigated the Minimum Spanning Tree problem in the context of EDO. They proved that for a small set including two solutions, diversity can be obtained in polynomial time. Neumann, Bossek, and Neumann [76] presented a method to compute diverse sets of solutions for submodular optimisation problems with uniform and knapsack constraints. They first introduced a sampling-based greedy algorithm to obtain diversity; then, they presented an EDO approach to improve the results. They also showed that their proposed EDO algorithm outperforms the greedy algorithm in submodular function problems. An EDO approach was recently adopted to effectively provide diverse sets of high-quality solutions for the detection and concealment of communication networks in large settings [79].

2.4 Combinatorial Problems

Any optimisation problem with a finite number of solutions can be defined as a combinatorial problem. This definition covers a broad range of real-world problems, from routing to scheduling or allocation problems. A combinatorial problem can be formally defined on a triple (S, Z, Ω) where S is the solution space, Z is the fitness function, and Ω is the set of constraints. As the definition above, solution space S must be discrete, and solutions can only take discrete value types.

In combinatorial optimisation problems, the objective is to compute a global optimum x^* with respect to the fitness function $Z(x)$ that complies with all problem constraints Ω. Finding global optima is not straightforward in many real-world problems due to its complex nature. Thus, the use of heuristics and metaheuristics, such as EAs, is very popular among researchers studying such problems. In this thesis, we study three well-known combinatorial problems, the TSP, the KP, and the TTP. We aim to introduce evolutionary-based methods inspired by EDO and QD frameworks to compute a diverse set of solutions for those problems. We seek to expand the literature on EDO and QD and our understanding of these problems.

2.4.1 Knapsack Problem

The KP is one of the best-known combinatorial problems. The KP is defined on a set of items I, where the cardinality of the set I is n ($|I| = n$) and each item i corresponds to a weight value w_i and a profit value p_i. Here, the goal is to find a selection of item $y = (y_1, y_2, \dots, y_n)$ that maximises the profit while the weight of selected items is constrained to a capacity W. Here, y is the characteristic vector of the selection of items where y_i is set to 0 if the item i is excluded from the selection y; otherwise, it is set to 1. Formally, we can formulate the KP as follows:

$$\text{Maximise } g(y) = \sum_{j=1}^{m} p_j y_j$$

$$\text{subject to } \sum_{j=1}^{m} w_j y_j \leq W.$$

We assume that all items have weights in $\{0, 1, 2, \cdots, W\}$ since any item violating this can be removed from the problem instance. The KP and its applications have been investigated extensively in the literature [100, 105]. Several algorithms have been proposed for solving the KP, such as dynamic programming [111] and meet-in-the-middle [51]. There are also several methods aiming for the approximation of the KP, including greedy approximation algorithm [22], and fully polynomial-time approximation scheme (FPTAS) [114]. here, we review the dynamic programming [111] since we utilse the algorithm in Chapter 5 and 6.

Dynamic Programming

Dynamic programming (DP) is a classical, exact approach to solving the KP to optimality, which is presented in [111]. The DP consists of a Table β including m rows from 1 to m and $W + 1$ columns from 0 to W. Here, items are processed in order of their indices. The lower the index, the sooner the item is to be processed. Let denote each table entry by $\beta_{i,w}$, which represents the maximum profit we can obtain when we only consider the first i items and use the knapsack capacity of w. For the first row of the table, we have $\beta_{1,w} = 0 \forall w < w_1$ and $\beta_{1,w} = p_1 \forall w \geq w_1$ since we cannot pack item 1 at a capacity lower than w_1. Let $j - 1$ be the predecessor of item j. Then, we can calculate the rest of the table from:

$$\beta_{j,w} = \max(\beta_{j-1,w}, p_j + \beta_{j-1,w-w_j})$$

Once we complete the table, the optimal profit can be found in the last cell of the table ($\beta_{m,W}$), where we take into account all the items and the full capacity of the knapsack. From that cell, we can go back and trace down all items that are included in the optimal solution and result in the maximum profit.

2.4.2 Traveling Salesperson Problem

The TSP is a well-known NP-hard combinatorial optimisation problem. The problem includes a salesperson who aims to find the shortest path possible to visit a number of cities exactly once and then return to the first city. The problem is defined on a complete graph $G = (V, E)$ where V is a set of nodes and E is a set of pairwise edges between the nodes, $e = (i, j) \in E$, each associated with a positive weight, $d(e)$.

In this research, we assume that the TSP instances are symmetric (i. e., $d(i, j) = d(j, i)\, \forall i, j \in V$). We denote by $n = |V|$ and $m = |E| = n(n-1)/2$ the cardinality of these sets. The objective is to compute a permutation $x : V \to V$ that minimises the following cost function:

$$c(x) = d(x(n), x(1)) + \sum_{i=1}^{n-1} d(x(i), x(i+1)).$$

This objective returns the distance solution x takes to finish the tour. The TSP has been extensively studied both practically and theoretically and has various real-world applications, such as drilling problem [44], school bus routing problem [4], the navigation satellite system [102]. Many other studies can be found in the literature that directly or indirectly are related to the TSP. The TSP also has been used for benchmarking and testing the performance of the algorithms. Therefore, Several algorithms can be found in literature capable of addressing the TSP efficiently [48, 58, 72, 121]. Here, we review two variation operators we use later in this thesis.

EAX

We consider EAX crossover [72] to generate new TSP tours. EAX is a highly performing TSP crossover known as one of the state-of-the-art operators in solving TSP. EAX has several variants; we incorporate the EAX-1AB due to its simplicity and efficiency. The EAX consists of three steps. Figure 2.1 depicts the three steps to implement the EAX-1Ab.

- AB-cycle: Generating one AB-cycle from the two parents by alternatively choosing edges from the first and second parents until a cycle is formed (Fig 2.1.2).

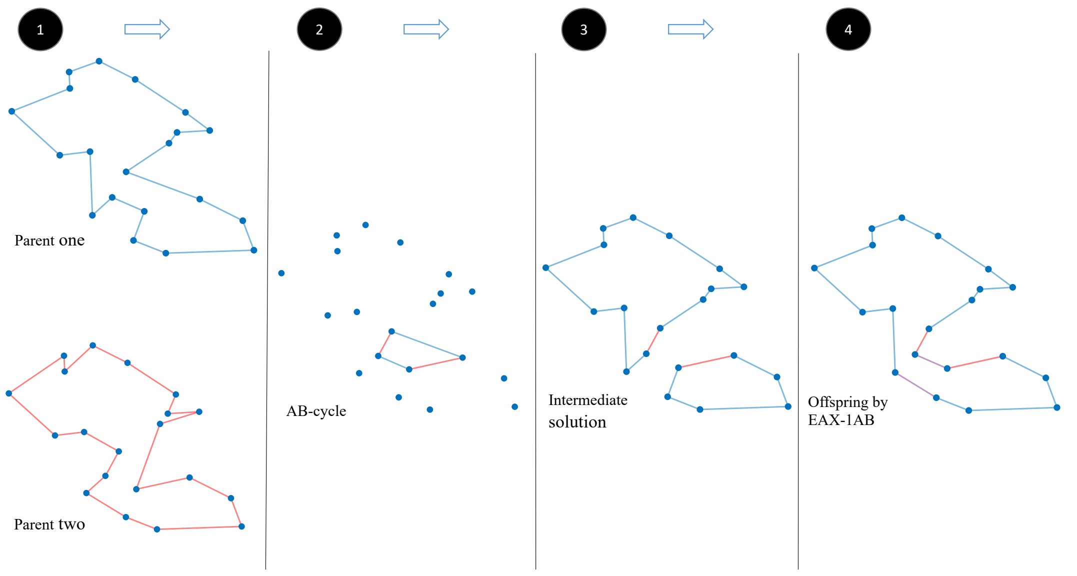


FIGURE 2.1. The representation of the steps to implement EAX.

- Intermediate Solution: Copying all edges of the first parent to the offspring; then removing the Ab-cycle's edges that belong to the first parent from the offspring, and adding the other edges of the AB-cycle to it (Fig 2.1.3).
- The Complete Tour: Connecting all sub-tours of the intermediate solution to form a complete tour (Fig 2.1.4).

To connect two sub-tours, we require discarding one edge from each sub-tour and adding two new edges to connect each end of the deleted edges. For Step 3, we start with the sub-tour (r) with the minimum edge number. Then, we select the 4-tuples of edges such that $\{e_1, e_2, e_3, e_4\} = \arg\min\{-d(e_1) - d(e_2) + d(e_3) + d(e_4)\}$ where $e_1 \in E(r)$ and $e_2 \in E(t) \setminus E(r)$. $E(t)$ and $E(r)$ represent the set of edges formed the intermediate solution t and sub-tour r, respectively. We refer interested readers to [72] for more details about the process of generating a new tour by the EAX. In this research, we also incorporate 2-OPT mutation into EAs to generate new TSP tours. In the following, we will review 2-OPT.

2-OPT

The 2-OPT [20] is originally a randomised neighbourhood search algorithm introduced for permutation problems. However, the operator that is used to change a solution and generate a new one has been extensively employed in other algorithms, such as EAs, as the variation operator. The operator is also called 2-OPT. The steps needed to implement the 2-OPT operator come as follows:

- Select two elements from the permutation uniformly at random.
- Swap the positions of the selected elements in the permutation.

- Reorder the elements between two selected elements in the backward direction.

2.4.3 Traveling Thief Problem

The TTP is formed by the integration of the travelling TSP and the KP. The TTP is defined on the graph G, same as TSP and a set of items I where items are scattered on the cities equally. Formally, every city i except the first one contains a set of items M_i (a subset of I). Same as KP, each item k located in the city i is associated with a profit p_{ik} and a weight w_{ik}. To ease the presentation, we do not use the double subscripts for the profits and weights in the following but refer directly to the items at one particular city when required.

The thief should visit all the cities exactly once, pick some items into the knapsack, and return to the first city. A rent R should be paid for the knapsack per time unit. The thief's speed non-linearly depends on the weight of the knapsack. In TTP, we aim to find a solution $t = (x, y)$ consisting of a tour x and a KP solution y (called a packing list in the context of TTP) that maximises the following profit function:

$$z(x,y) = g(y) - R\left(\frac{d_{x_n x_1}}{\nu_{max} - \nu W_{x_n}} + \sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{\nu_{max} - \nu W_{x_i}}\right)$$

$$\text{subject to } \sum_{j=1}^{m} w_j y_j \leq W.$$

Here, $\nu_{\max}$ and $\nu_{\min}$ are the maximal and minimal travelling speed, $\nu = \frac{\nu_{max} - \nu_{min}}{W}$ is a constant, and W_{x_i} is the cumulative weight of the items collected from the start of the tour up to city x_i.

TTP was introduced in 2013 by Bonyadi, Michalewicz, and Barone [9]. TTP is the combination of the classical TSP and the KP. Both TSP and KP are well-known, well-studied combinatorial problems. In a nutshell, they integrate the TSP and the KP so that the travelling cost between two cities depends not only on the distance between the cities but also on the weight of the items collected so far.

In recent years, several solution approaches have been introduced to TTP. This includes algorithms based on co-evolutionary strategies [10, 123], local search heuristics [62, 93], profit-based heuristic that modifies tours based on the items' value [73], simulated annealing [124], swarm intelligence approaches [115, 128]. Furthermore, an adaptive surrogate model was proposed in [74] to filter our non-promising tours. Exact methods based on dynamic programming have been introduced in [119], but they are limited to solving only small instances. Moreover, Wuijts and Thierens [120] investigated the fitness landscape of some small instances of TTP. They studied local search and genetic algorithms using a wide range of operators such as 2-opt, insertion, EAX, and PMX. They conclude that genetic algorithms using EAX can outperform the other algorithms under investigation in those instances.

In cases where the tour is fixed, and the packing list is to be optimised, the problem is referred to as packing while travelling (PWT) [93] in the literature. Neumann et al. [81] introduced a DP algorithm to solve the PWT problem to optimality.

Dynamic Programming

DP is a classical approach to solving the KP. Here, we employ the DP introduced in [81] to solve the PWT problem. The DP includes a table β consisting of m rows from 1 to m and $W + 1$ columns from 0 to W. In the DP, items are processed in the order that their corresponding node appears in the tour. For example, I_i is processed sooner than I_j if the node to which I_i belongs is visited sooner than the node of I_j. If two items belong to the same node, they are processed according to their indices. The entry $\beta_{i,j}$ represents the maximal profit that the thief can obtain among all combinations of items I_k with $I_k \preceq I_i$, bringing about the weight exactly equal to j. If no combinations lead to the weight j, $\beta_{i,j}$ is set to $-\infty$.

Let denote the profit of the empty set by $B(\emptyset)$, which is equal to the travelling cost with an empty knapsack. Moreover, we denote the profit by $B(I_1)$ when only item I_1 is collected. Thus, for the first item I_1 (the first row of the table β) based on the order aforementioned, we have:

$$\beta_{1,0} = B(\emptyset), \quad \beta_{1,w_1} = B(I_1), \quad \beta_{1,j} = -\infty, \forall j \notin \{0, w_1\}$$

Let denote the predecessor of I_i by I_{i-1}. For the rest of the table, each entry $\beta_{i,j}$ can be computed from $\max(\beta_{i-1,j}, T)$, where

$$T = \beta_{i-1,j-w_i} + p_i - R \sum_{l=1}^{n} d_l \left(\frac{1}{\nu_{max} - \nu j} - \frac{1}{\nu_{max} - \nu j - w_i} \right)$$

The $\max_j \beta_{m,j}$ is reported as the optimal profit that the thief can gain from the given tour. The interested readers are referred to [81], which analysed the runtime of the DP.

2.4.4 Boolean Satisfiability Problem

The boolean satisfiability problem (SAT) consists of determining the existence of an assignment (also called model, interpretation, or solution) satisfying a Boolean formula. A Boolean formula is several literals combined by logical connectives, AND ($\wedge$), and OR ($\vee$), and a literal is a Boolean variable or a negation of a variable ($\neg$). A formula that is formed by the conjunction of a number of clauses (a disjunction of literals) is in conjunctive normal form (CNF). A formula in CNF is satisfiable if there is at least one assignment of the variables such that the formula evaluates to true. In other words, a given CNF formula Φ is true if an assignment x satisfies all clauses in Φ; otherwise, Φ is false.

Several efficient approaches have been developed for SAT in recent years, like conflict-driven clause-learning (CDCL) [106] and the variable state independent decaying sum (VSIDS) branching heuristic [66]. One very versatile solver incorporating those heuristics is minisat [32]. Minisat has been widely adopted and used as the benchmark solver in the literature. To the best of our knowledge, Nadel [68] is the only study focusing on the diversity of SAT solutions. The maximum satisfiability problem (MAX-SAT) is a generalisation of the SAT, where the problem is defined as the determination of the maximum number of clauses that can be made true by an assignment. Since MAX-SAT can fit into the category of optimisation problems, it has been studied in the context of EC more frequently compared to SAT.

2.5 Methodology

In this thesis, we first define diversity for the optimisation problems under investigation; then, we design algorithms to compute a diverse set of solutions for the problems by taking into account the problems' characteristics. Afterwards, we empirically scrutinise and investigate the algorithms' performance by designing appropriate experiments and comparing the algorithms with baselines in the literature. We compare the mean of several independent runs of the algorithms, and we also use statistical hypothesis testing to examine the significance of differences in the results.

Statistical hypothesis testing refers to determining if the obtained data supports a hypothesis. This kind of test can help us to make probabilistic statements about the results we obtain from the EAs. Here, we utilise the Kruskal–Wallis at significance level 5% test with Bonferroni correction. The Kruskal-Wallis test is a method to determine whether two or more independent samples belong to the same distribution, and Bonferroni correction is a method to counteract testing multiple hypotheses. Interested readers are referred to [98] for more details about the statistical tests. Moreover, we conduct other experimental investigations when they are appropriate. For instance, we look into the trajectory of the algorithms or design simulations to test the diversity of solutions.

In this thesis, the experimental setting is chosen based on several factors, such as the problems under investigation, the proposed algorithms, the aims of the experiments, and the computational budget. Since these factors are not static throughout the thesis, the experiment setting changes to adopt the requirements. We specify the setting at the beginning of the experimental sections.

2.6 Conclusions

This chapter gave an introduction to combinatorial optimisation problems and the solution approaches to address them, including well-known bio-inspired meta-heuristics in the literature, such as EA. We also provided a review of the studies investigating the diversity of solutions and aimed for the construction of a set of diverse, high-quality solutions for the problem at hand. In Ec, diversity is seen as niching in solutions' fitness or behavioural landscape or structural properties.

Computing diverse solutions has received increasing attention among researchers in the EC community. The benefits of diverse solutions have been highlighted in several studies, including in [79, 80]. The invaluable knowledge in solution space, robustness against imperfect modelling, and freedom of choice, among others, can be cited as the advantages of a diverse set of solutions compared to a single optimal solution.

Chapter 3

Entropy-based diversity in the Traveling Salesperson Problem

3.1 Introduction

Several algorithms have been introduced to find an (approximately) optimal solution for the TSP [48, 58, 72, 121], but only a few studies focused on maximising the diversity of solutions (see, e.g., Do et al. [24]). The diversity measures adopted in Do et al. [24] do not consider dependencies between nodes in the TSP but focus on the frequency of the occurrence of edges in the EAs' population. This is while the nodes in a TSP tour are strongly correlated to each other [70] (we will explain this matter further in Section 3.2). Therefore, we incorporate a diversity measure based on entropy into EDO for the TSP in this study. This measure explicitly addresses the dependency between nodes in the TSP tours.

We examine the diversity measure's theoretical properties and determine characteristics that a maximally/minimally diverse set of tours should possess. Besides, we propose a Mixed-Integer Programming (MIP) formulation of the considered diversity problem and solve it with an exact solver to a) support the theoretical proofs and b) use it as a baseline for experimentation. Then, we introduce the biased 2-OPT mutation, which mainly focuses on more frequent components in the population and aims to decrease their frequency to increase diversity. We perform an extensive experimental study in the unconstrained case (no quality criterion) and the constrained case with (un)biased 2-OPT mutation operators. Our results indicate a clear advantage of the entropy-based driven EA compared to EAs based on the distance-based diversity measures introduced by Do et al. [24]. The results also show that biased 2-OPT brings faster convergence, especially in unconstrained diversity optimisation.

Furthermore, previous studies have overlooked a potential impact of the $(\mu + \lambda)$ EA where $\lambda \geq 2$ due to the complexity of diversity calculations. Here, μ and λ denote the size of the main population and offspring pool. This complexity is rooted in the fact that diversity is a measure calculated for a set of solutions, and thus, finding the best μ solutions from a pool of $\mu + \lambda$ is a complex task. In fact, there are $\binom{\mu+\lambda}{\mu}$ possible sets, and an increase in λ leads to a substantial increase in the complexity of the subset selection problem at hand. To overcome this challenge, we introduce multiple selection methods, including greedy, tournament, and EA-based selection, which not only deal with the increased complexity but also examine the impact of $(\mu + \lambda)$ EA where $\lambda \geq 2$ on the results. Moreover, we calculate how many diversity evaluations each selection method requires in each generation. We examine the methods empirically and compare them with the conventional $(\mu+1)$ EA for both constrained and unconstrained diversity optimisation. The investigation indicates that the $(\mu+\lambda)$ EA version using the EA-based selection outperforms the conventional $(\mu + 1)$ EA in both unconstrained and constrained optimisation.

The work of this chapter is based on a conference paper [83] presented at the genetic and evolutionary computation conference (GECCO 2021) and its extended version that is submitted to evolutionary computation journal. The chapter is structured as follows: In Section 3.2, we define the TSP problem and diversity and determine the maximum and minimum values for the high-order entropy measure in 3.3. Section 3.4 presents a MIP formulation, while Section 3.5 proposes an EA and the (un)biased 2-OPT mutation. Section 3.6 empirically compares the MIP and EA, biased and unbiased mutation, and entropy-based and distance-based diversity measures. Sections 3.7 and 3.8 introduce and experimentally compare three different sub-set selection methods and the conventional $(\mu + 1)$EA. Finally, we conclude with some remarks and promising future research perspectives.

3.2 Maximising Diversity in TSP

In this chapter, we examine the TSP in the context of EDO. TSP is formally defined in 2.4.2. Given a TSP instance G, let OPT be the cost of the optimal tour for G and $\alpha > 0$ be a predefined parameter. The objective is to compute a diverse set of tours where a) the diversity value of the population is maximised in terms of a given diversity measure; b) all individuals comply with a maximum cost, i. e., $c(p_i) \leq (1+\alpha)OPT, \forall p_i \in P$. In this chapter, p represents a solution in the population P. Here, the goal is to maximise the diversity of the set of solutions subject to the quality constraint. Maximising the diversity of tours provides us with valuable information on the solution space around the optimal tour. It can indicate which edges are irreplaceable or complex to replace if we aim to stay within the quality threshold. Moreover, it enables decision-makers to choose between different tours; they may decide to visit a city earlier than another or avoid an edge if provided with various alternatives with reasonable costs.

Recently, Do et al. [24] studied EDO on TSP for the first time. The authors tailored two edge-based diversity measures, edge diversity (ED) and pairwise distance (PD), towards the TSP. ED measures the diversity based on the equalisation of the frequency of edges in the population. For this purpose, they used the notion of genotypic diversity [127] defined as the mean of pairwise distances:

$$ED(P) = \sum_{p \in P} \sum_{q \in P} |E(p) \setminus E(q)|,$$

where $E(p)$ is the set of edges of $p = (p(1), \dots, p(n))$.

$$(i.\,e., E(p) = \{(p(1), p(2)), (p(2), p(1)) \dots, (p(n), p(1)), (p(1), p(n))\})$$

On the other hand, PD is defined as

$$PD(P) = \frac{1}{n\mu} \sum_{p \in P} \min_{q \in P \setminus \{p\}} |E(p) \setminus E(q)|.$$

and emphasises uniform pairwise edge distances. PD is closely aligned with the diversity measure discussed by Wang, Jin, and Yao [117]. For the sake of brevity, we refer the reader to Do et al. [24] for further details.

One disadvantage of both ED and PD is that the dependency of the occurrence of nodes in a tour is not considered. This is while the occurrence of nodes in a tour is significantly dependent on each other in the TSP. Here, we show a tour as a permutation p consisting n decision variables $p(i)$ representing the i-th node visited in the tour. For instance, if we construct a tour manually, the next node we choose (the value of $p(i+1)$) is heavily dependent on the current node ($p(i)$), and all already visited nodes [70]. This is because we must not choose any already visited node. This issue can result in an inaccurate diversity evaluation. We employ an entropy-based diversity measure introduced by Nagata [70], termed high-order entropy, to resolve this issue. The measure considers the sequence of k nodes ($k-1$ edges) in tours instead of focusing on edges one by one. Nagata [70] showed that the high-order entropy measure outperforms the independent entropy measure in terms of preventing premature convergence.

3.3 High-Order Entropy Measure

In the high-order entropy measure, the sequence of $2 \leq k \leq n$ nodes ($k-1$ edges) in tours is the feature intended to be diversified. Let $s = \{v_1, \dots, v_k\}, v_i \in V$ for $i = 1, \dots, k$ be a segment consisting of k nodes. Then, its contribution to the overall entropy of the population P is given as

$$h(s) = -\left(\frac{f(s)}{2n\mu}\right) \cdot \ln\left(\frac{f(s)}{2n\mu}\right),$$

where $f(s)$ is the absolute number of occurrences of segment s in P. Note that $2n\mu$ is the total number of occurrences of all segments in a population of size μ since we are

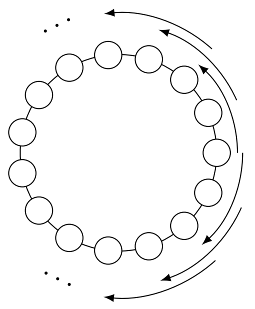


FIGURE 3.1. Illustration of building all segments of length $k = 3$ for a TSP tour.

able to traverse each tour in both directions. Each tour contains exactly $2n$ different segments (see Figure 3.1 for an example). In the following, it is sometimes useful to represent a segment by means of its set of edges. For instance, $s = \{s(1), s(2), s(3)\}$ can be also written as $E(s) = \{(s(1), s(2)), (s(2), s(3))\}$. Summing over all segments included in the population P, the entropy of P is defined as

$$H(P) = \sum_{s \in P} h(s).$$

Let $S = \{s_1, \ldots, s_u\}$ be the set of all possible segments of k nodes for a given TSP instance G, and let $u = |S| = \frac{n!}{(n-k)!}$ denote the cardinality of S. We sort the segments according to the number of their occurrences within P in an increasing order to obtain the vector $F(P) = (f(s_1), \ldots, f(s_u))$. It means that $f(s_1) \leq f(s_2) \leq \ldots \leq f(s_u)$. We define $f_{\min} = f(s_1)$, $f_{\max} = f(s_u)$, and $C = f_{\max} - f_{\min}$ where $f_{\min}$ and $f_{\max}$ are the smallest and the largest number of occurrences of segments in P, respectively. Intuitively, a maximally diverse population would have all $f(s_i)$ almost equalised. We will use $F(P)$ later to analyse whether a given population P has the maximum achievable entropy.

3.3.1 Maximum High-Order Entropy

Next, we aim to determine the characteristics of an ideal set of tours having the maximum high-order entropy value $H_{\max}$ for a given TSP instance.

Knowing $H_{\max}$ is important for two main reasons: a) it enables us to have a better understanding of an algorithm's performance by comparing the entropy of the final population with $H_{\max}$ and b) it allows us to use it as a termination criterion for an EA in the course of experimental evaluation with a fixed-target perspective.

Lemma 1. *Let P_2 be a population obtained from a population P_1 by decreasing $f_{\max}$ and increasing $f_{\min}$ by one unit each. If $C \geq 2$, then we have $H(P_2) > H(P_1)$.*

In order to show Lemma 1, we work under the assumption that $C \geq 2$ ($C = f_{\max} - f_{\min}$) and show that $H(P_2) - H(P_1) > 0$ holds. We show that $H(P_2) - H(P_1)$ is monotonically decreasing in $f_{\max}$ and $\lim_{f_{\max}\to+\infty} H(P_2) - H(P_1)$ converges to zero. This implies that Lemma 1 is true. The differences in P_1 and P_2 can be summarised in $f_{\min}$ and $f_{\max}$ where $f_{\max}$ decreased and $f_{\min}$ increased by one unit each in P_2. The number of occurrences of other segments is the same in both populations. Also, we have $f_{\max} = f_{\min} + C$. To simplify the following presentation, we use $f = f_{\max}$ and $f_{\min} = f - C$. Thus, we have:

$$\begin{aligned} H(P_2) - H(P_1) = &-\frac{f-1}{2n\mu}\ln\left(\frac{f-1}{2n\mu}\right) - \frac{f-C+1}{2n\mu}\ln\left(\frac{f-C+1}{2n\mu}\right) \\ &+\frac{f}{2n\mu}\ln\left(\frac{f}{2n\mu}\right) + \frac{f-C}{2n\mu}\ln\left(\frac{f-C}{2n\mu}\right) \end{aligned}$$

We now show that $H(P_2) - H(P_1)$ is monotonically decreasing in f if and only if $C \geq 2$.

Lemma 2. *If $C \geq 2$ then $H(P_2) - H(P_1)$ is monotonically decreasing in f.*

Proof. To prove $H(P_2)-H(P_1)$ is monotonically decreasing, we show that $\frac{d(H(P_2)-H(P_1))}{df} < 0$. We have

$$\begin{aligned} & \frac{d(H(P_2) - H(P_1))}{df} < 0 \\ \Leftrightarrow\ & \frac{1}{2n\mu}\left(\ln\left(\frac{f}{2n\mu}\right) + \ln\left(\frac{f-C}{2n\mu}\right)\right) - \\ & \frac{1}{2n\mu}\left(\ln\left(\frac{f-1}{2n\mu}\right) + \ln\left(\frac{f-C+1}{2n\mu}\right)\right) < 0 \\ \Leftrightarrow\ & \ln\frac{f(f-C)}{(f-1)(f-C+1)} < 0 \\ \Leftrightarrow\ & \frac{f(f-C)}{(f-1)(f-C+1)} < 1 \\ \Leftrightarrow\ & (f-1)(f-C+1) > f(f-C) \\ \Leftrightarrow\ & f^2 - f\cdot C + C - 1 > f^2 - f\cdot C \end{aligned}$$

The last expression holds as $C \geq 2$, which completes the proof. □

Owing to Lemma 2, if $H(P_2) - H(P_1)$ is still positive for an extremely large f, it is positive for all smaller values of f. We now investigate f approaching to $+\infty$.

Lemma 3. *$H(P_2) - H(P_1) > 0$ holds for any fixed population size μ and $C \geq 2$.*

Proof. As one can notice, f is bounded by μ. that means f can approaches to $+\infty$, only if μ approaches to $+\infty$ as well. Thus, we investigate $H(P_2) - H(P_1)$ in the most extreme case where μ and $f \to +\infty$.

$$\begin{aligned} H(P_2) - H(P_1) \quad \Leftrightarrow \quad & \left(\frac{f-C}{2n\mu}\right)\ln\left(\frac{f-C}{2n\mu}\right) - \left(\frac{f-1}{2n\mu}\right)\ln\left(\frac{f-1}{2n\mu}\right) + \\ & \left(\frac{f}{2n\mu}\right)\ln\left(\frac{f}{2n\mu}\right) - \left(\frac{f-C+1}{2n\mu}\right)\ln\left(\frac{f-C+1}{2n\mu}\right) \end{aligned}$$

We compute the limit for μ and $f \to +\infty$ by applying L'Hopital's rule and have:

$$\begin{aligned} & \left(\frac{1}{2n}\right)\ln\left(\frac{1}{2n}\right) - \left(\frac{1}{2n}\right)\ln\left(\frac{1}{2n}\right) + \\ & \left(\frac{1}{2n}\right)\ln\left(\frac{1}{2n}\right) - \left(\frac{1}{2n}\right)\ln\left(\frac{1}{2n}\right) = 0 \end{aligned}$$

The last expression shows that $H(P_2) - H(P_1)$ converges to 0 if $f \to +\infty$. We have $f \le \mu$ and using Lemma 2, this implies that $H(P_2) - H(P_1) > 0$ for any fixed μ if $C \ge 2$. □

Theorem 1. *For every complete graph with n nodes and every population size $\mu \ge 2$, the entropy of a population P with μ individuals is maximum if and only if C is equal to zero or one.*

Proof. Lemma 1 shows that a population's entropy can be increased as long as $C \ge 2$. Therefore, C should be equal to 0 or 1 to have a maximum entropy population. □

To have C to 0 or 1, the number of occurrences of all possible segments should be equalised. For every TSP instance, there are u possible segments and $2n\mu$ occurrences of all segments for every population. The optimal value of $f_{\min}$ is equal to $[\frac{2n\mu}{u}]$. Let $f^*_{\min}$ and C^* be the values of $f_{\min}$ and C in an optimal population. It should be noted that based on the Pigeonhole principle if $\frac{2n\mu}{u}$ is an integer, $C \in \{0, 2, 3, \ldots, u\}$ and $C^* = 0$; otherwise, $C \in \{1, 2, \ldots, u\}$ and $C^* = 1$. In other words, C can get only one of the values of 0 or 1 depending on the parameters of the problem, such as the size of the population, segments, and TSP instances. All in all, $(f^*_{\min} + 1)u - (2n\mu)$ segments occur $f^*_{\min}$ times in an optimal population whereby, the number of occurrences of the other segments is equal to $f^*_{\min} + C^* = f^*_{\max}$.

$$\begin{aligned} H_{\max} = & -((2n\mu) - (f^*_{\min} \cdot u))\left(\frac{f^*_{\max}}{2n\mu}\right)\ln\left(\frac{f^*_{\max}}{2n\mu}\right) \\ & - ((f^*_{\min} + 1)u - (2n\mu))\left(\frac{f^*_{\min}}{2n\mu}\right)\ln\left(\frac{f^*_{\min}}{2n\mu}\right) \end{aligned} \tag{3.1}$$

Note that the entropy of any set of TSP tours is always greater than zero. This is because no segments are allowed to occur within a tour more than once. In the worst-case scenario where a population consists of μ copies of a single tour, we have $2n$

different segments with the number of occurrences μ. We can determine the entropy value of a population with such characteristics from:

$$H_{\text{min}} = -2n \left(\frac{1}{2n} \ln \left(\frac{1}{2n} \right) \right) = \ln(2n) \tag{3.2}$$

3.4 Mixed-Integer Programming Formulation

This section presents a MIP formulation for the considered problem. Solving the proposed MIP with an exact solver such as the Cplex [19] can help us to observe whether the solver obtains the same results as Formula 3.1. Also, it would provide us with a baseline to investigate the performance of other algorithms. The objective function is formulated as follows:

$$H(P) = \sum_{s \in P} - \left(\frac{f(s)}{2n\mu} \right) \ln \left(\frac{f(s)}{2n\mu} \right) \rightarrow \max! \tag{3.3}$$

where $f(s), s = \{v_i, \dots, v_q\}$, is calculated from

$$f(s) = \sum_{p \in P} x^p_{ij} \cdots x^p_{tq} + \sum_{p \in P} x^p_{ji} \cdots x^p_{qt} \tag{3.4}$$

Here, x^p_{ij} is a binary variable; it is set to 1 if edge $e = (i, j)$ is included in tour p; otherwise, it is equal to zero. For example, if $s = (v_3, v_5, v_2, v_1)$, $f(s) = \sum_{p \in P}(x^p_{35} \cdot x^p_{52} \cdot x^p_{21}) + \sum_{p \in P}(x^p_{12} \cdot x^p_{25} \cdot x^p_{53})$. The maximisation of the objective function in Eq. 3.3 is subject to the following constraints:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} d(i,j) x^p_{ij} \leq (1+\alpha) \cdot OPT, \quad \forall p \in P \tag{3.5}$$

$$\sum_{i=1, i \neq j}^{n} x^p_{ij} = 1, \quad \forall j \in V, p \in P \tag{3.6}$$

$$\sum_{j=1, i \neq j}^{n} x^p_{ij} = 1, \quad \forall i \in V, p \in P \tag{3.7}$$

$$w^p_i - w^p_j + n x^p_{ij} \leqslant n - 1, \forall i, j \in V, i \neq j, p \in P \tag{3.8}$$

$$w^p_i \leqslant n - 1, \quad \forall i \in \{2, \dots, n\}, p \in P \tag{3.9}$$

$$x^p_{ij} \in \{0, 1\}, w_i \geq 0, \forall i, j \in V, p \in P. \tag{3.10}$$

Here, w^p_i is a positive integer showing the position of node i in the tour p. Equation 3.5 makes sure that all solutions satisfy a minimal quality with respect to tour length. Equations 3.6 and 3.7 guarantee that all nodes are visited exactly once in each tour, while Equations 3.8 and 3.9 prevent the creation of sub-tours as proposed by Miller, Tucker, and Zemlin [65]. The objective function (Eq. 3.3) should be linearised to use the MIP solvers.

Linearisation

In Section 3.3.1, we showed that the entropy value of a population P is maximum if and only if $C = f_{\max} - f_{\min} \in \{0, 1\}$. In other words, Equation 3.3 is maximised if and only if C is set to zero or one. Therefore, We can replace the objective function with

$$C = f_{\max} - f_{\min} \rightarrow \min! \tag{3.11}$$

Here, $f_{\min}$ and $f_{\max}$ are dependent on x_{ij}^p. Thus, the correlation between these two variables and the other MIP variables should be explicitly defined in the MIP formulation before using Equation 3.11 as the MIP's objective function. For this purpose, we need to add new constraints and variables.

$$y_{ij\cdots s}^p \geq x_{ij}^p + \ldots + x_{tq}^p - k + 2, \quad \forall i, \ldots, q \in V, p \in P \tag{3.12}$$

$$y_{ij\ldots s}^p \geq x_{qt}^p + \cdots + x_{ji}^p - k + 2, \quad \forall i, \ldots, q \in V, p \in P \tag{3.13}$$

$$\sum_i \sum_j \cdots \sum_t \sum_p y_{ij\cdots sq}^p \leq 2n\mu \tag{3.14}$$

$$f_{\max} \geq \sum_{p \in P} y_{ij\cdots tq}^p, \quad \forall i, \ldots, q \in V \tag{3.15}$$

$$f_{\min} \leq \sum_{p \in P} y_{ij\cdots tq}^p, \quad \forall i, \ldots, q \in V \tag{3.16}$$

Here, $y_{ij\cdots tq}^p$ is a binary variable set to 1 if segment $s = \{v_i, \vdots, v_q\}$ or $s' = \{v_q, \vdots v_i\}$ is included in tour p. For example, if the tour p includes either of the segments $s = \{v_3, v_5, v_2, v_1\}$ (i. e. $E(s) = \{(3, 5), (5, 2), (2, 1)\}$) or $s' = \{v_1, v_2, v_5, v_3\}$, both y_{3521}^p and y_{1253}^p are set to 1. Note that segments s and s' are identical since we can traverse a tour in both directions. Equations 3.12 and 3.13 ensure that $y_{ij\cdots q}^p$ is set to 1 if segment s is included in tour p. Equation 3.14 guarantees that $y_{ij\cdots q}^p$ is equal to zero if the segment s is not included in the tour p. Finally, Equations 3.15 and 3.16 determine $f_{\min}$ and $f_{\max}$. Moreover, $f(s)$ can be calculated from summing up $y_{ij\cdots q}^p$ over p. In the final MIP formulation, Equation 3.11 serves as the objective function subject to the constraints [3.5-3.10] and [3.12-3.16].

3.5 Entropy-based Evolutionary Diversity Optimisation

We introduce an EA to address EDO for TSP tours (see Algorithm 5 for an outline). The algorithm is initialised with a population P consisting of μ copies of an optimal tour/permutation for the given TSP instance. A broad range of successful algorithms

Algorithm 5 Diversity maximising EA

```
1: Initialise the population $P$ with $\mu$ copies of the optimal TSP tours.
2: Choose $p \in P$ uniformly at random and generate a offspring $p'$ by mutation
3: if $c(p') \leq (1+\alpha) \cdot OPT$ and $H(P \setminus \{p\} \cup \{p'\}) \geq H(P)$ then
4:     Replace $P$ with $P'$.
5: Repeat steps 2 to 6 until a termination criterion is reached.
```

is proposed in the literature to find the optimal tour in TSP, such as Concorde by Applegate et al. [6]. Moreover, the optimal tours have been provided for most benchmark instances in the well-known TSPlib [97]. Next, a parent p is selected uniformly at random, and mutation operators generate offspring individuals, p'. Then, p is replaced with p' if p' satisfies the quality constraint and $P \setminus \{p\} \cup \{p'\}$ has a higher diversity than P. From the entire population, we solely consider parents for survival selection to increase time efficiency. We will discuss that the exclusion of the rest of the population does not affect the results significantly. These steps are repeated until a termination criterion is met.

3.5.1 Biased 2-OPT

We present two biased versions of the 2-OPT mutation operator, which is widely used in TSP. In the classic 2-OPT, two edges are randomly selected and swapped, and the nodes between them are reversed. However, this operator may not be effective in maintaining diversity in the population. Our biased versions aim to address this by incorporating a bias towards high-frequency segments in the population. In the *normalised biased 2-OPT*, there is competition among the high-frequency segments, where each segment's likelihood is proportional to its frequency. On the other hand, the *absolute biased 2-OPT* only selects the segment with the highest frequency. Since the latter version focuses only on the most frequent segments, it is useful in unconstrained diversity optimisation where the quality constraint is not a concern. In constrained diversity optimisation, the normalised biased version is more appropriate. By considering the frequency of each segment, this version increases the probability of generating offspring that meet the quality criterion while still increasing diversity in the population. Overall, these biased versions of 2-OPT can improve the performance of TSP algorithms in maintaining diversity and exploring new solutions.

Owing to the parent and the offspring's similarity, the algorithm compares the offspring with its parent rather than the entire population. All versions of 2-OPT change solely two edges of a parent. So, the population's entropy is likely to decrease if both parent and offspring remain in the population, especially in unconstrained diversity optimisation. In constrained diversity optimisation, it can be beneficial to compare an offspring to the entire population, but it increases the computational costs. We introduce different selection methods that allow us to generate a pool of offspring and replace more than one individual in each generation in Section 3.7 and 3.8.

TABLE 3.1. Comparison of the entropy of final populations obtained by Cplex and the EA (symbols O and N indicate whether Cplex converged within the given time-bound).

		n = 5			n = 10			n = 15			n = 20		
μ	k	**ENT**	**Cplex**	$H_{\max}$	**ENT**	**Cplex**	$H_{\max}$	**ENT**	**Cplex**	$H_{\max}$	**ENT**	**Cplex**	$H_{\max}$
6	2	3.00	3.00 (O)	3.00	4.44	4.44 (O)	4.44	5.19	5.19 (O)	5.19	5.48	5.48 (O)	5.48
6	3	4.09	4.09 (O)	4.09	4.79	4.79 (O)	4.79	5.19	5.19 (O)	5.19	5.48	5.48 (O)	5.48
12	2	3.00	3.00 (O)	3.00	4.48	4.48 (O)	4.48	5.31	5.31 (O)	5.31	5.88	5.88 (O)	5.88
12	3	4.09	4.09 (O)	4.09	5.48	5.48 (O)	5.48	5.89	5.89 (O)	5.89	6.17	6.17 (O)	6.17
24	2	3.00	3.00 (O)	3.00	4.50	4.50 (O)	4.50	5.34	5.33 (O)	5.34	5.92	5.91 (N)	5.92
24	3	4.09	4.09 (O)	4.09	6.17	-	6.17	6.58	-	6.58	6.87	-	6.87

3.6 Experimental Investigation

In this section, we conduct a series of experiments to evaluate the suitability of the proposed algorithm and diversity measure while Section 3.8 is dedicated to investigation of survival selection methods. Here, the experiments are classified into three parts. First, we examine the algorithm's results to make sure that a) the considered survival selection does not affect the entropy of the final population by comparing the results with an EA including the entire population in the survival selection procedure and b) the results obtained from our EA are consistent with the results of the Cplex solver and $H_{\max}$ (see Eq. 3.1). Subsections 3.6.2 and 3.6.3 are dedicated to comparing the introduced EA and the EAs based on PD and ED by Do et al. [24] in unconstrained diversity optimisation and constrained diversity optimisation, respectively.

3.6.1 Validation of the Proposed EA

This subsection scrutinises the performance of the introduced EA. Initially, we examine the survival selection method by comparing the proposed approach to a more general version where the offspring is compared to the entire population. Later, we draw an analogy between the EA and an exact method.

Survival Selection Procedure

As mentioned, it is more efficient to compare the offspring with the parent than the entire population, especially in unconstrained optimisation. Here, we analyse the algorithm's survival selection and compare it with the same algorithm where the offspring is compared with all individuals in the population. All combinations of $n = \{25, 50\}$, $\mu = \{12, 20, 50\}$, and $k = \{2, 3, 4\}$ are subject to experimentation. Due to the relaxation of the quality constraint, we consider complete graphs where the edges' weights are all equal to one as TSP instances for unconstrained diversity optimisation. The termination criterion is reaching the limit of 100 000 generated offspring. The results show no significant differences in the mean of entropy values over all the cases. The observation is confirmed with the Kruskal-Wallis test at significance level 95% and the Bonferroni correction method. Note that in the constrained version, if a tight threshold is considered, it is beneficial to have the entire population included in survival selection. Also, it can sometimes be beneficial to increase the offspring pool and change more than one individual in the population to escape local optima, although it increases the problem complexity. Thus, we introduce different selection methods where it is possible to change more than one solution in each generation in Section 3.7.

Comparison between the exact solver and the proposed EA

In this section, we focus on unconstrained diversity optimisation to examine the results obtained from solving the MIP formulation using the Cplex solver. We do not consider constrained optimisation for two reasons. Firstly, our main objective in using an exact solver such as Cplex is to support the formula for $H_{\max}$, which can serve as a baseline for larger instances where the Cplex solver is unable to solve the problem within a bounded time. However, imposing quality constraints can remove parts of the solution space to which $H_{\max}$ belongs, making it difficult to verify the formula in constrained diversity optimisation. Secondly, the Cplex solver is not suitable for solving medium or large instances, even in unconstrained diversity optimisation. Therefore, investigating small instances alone serves no purpose. The experiments take place on all combinations of $\mu \in \{6, 12, 24\}$, $n \in \{5, 10, 15, 20\}$ and $k \in \{2, 3\}$. A time-bound of 24 hours is considered for the Cplex solver. The results are summarised in Table 3.1. Note that the MIP formulation's objective function is to minimise $f_{\max} - f_{\min}$, while the EA uses the entropy value as the fitness function.

In Table 3.1, O and N represent the capability and incapability of the Cplex solver in converging to the global optimum within the time-bound, respectively. As Table 3.1 indicates, there are cases where Cplex cannot solve instances to the optimal value ($n = 20$, $\mu = 24$, and $k = 2$), and cases where Cplex cannot find a feasible solution, $n \in \{10, 15, 20\}$, $\mu = 24$, and $k = 3$. These cases highlight the need for a time-efficient algorithm. More importantly, Table 3.1 shows that if the Cplex solver finds the optimal solution, the proposed EA converges to a population with the same entropy, which is consistent with the $H_{\max}$ 3.1. The fact that both Cplex and the EA converged to $H_{\max}$ supports Equation 3.1. Therefore, we can use $H_{\max}$ as another termination criterion of the introduced EA and the baseline for further experimental investigation. Note that the introduced EA converges in less than two minutes.

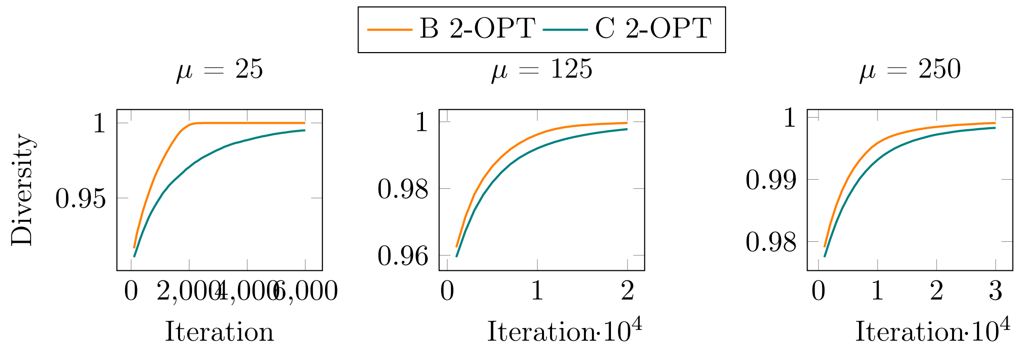


FIGURE 3.2. Comparison between convergence pace of biased (orange) and classic 2-OPT (green).

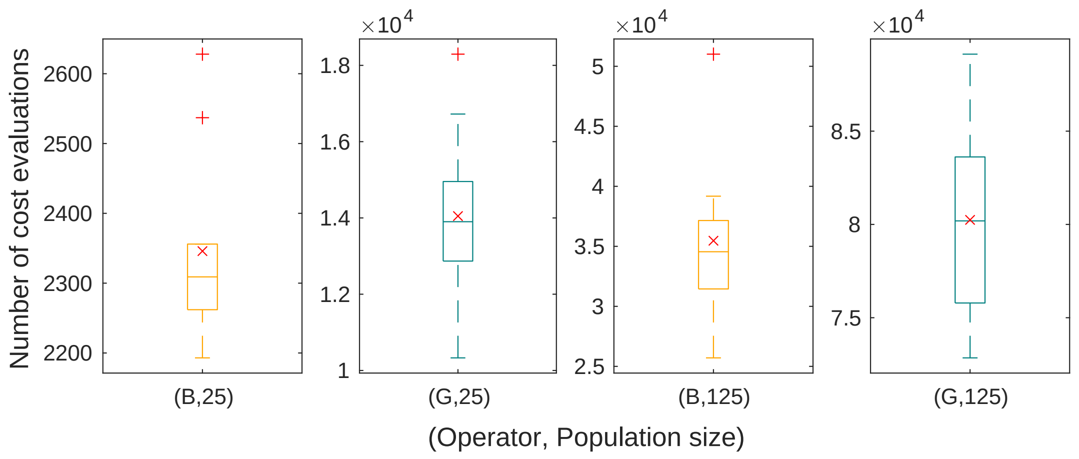


FIGURE 3.3. The number evaluations in reaching $H_{\max}$ for the biased (orange) and the classic 2-OPT (green). The red $\times$ and $+$ show the mean of the results and the outliers, respectively.

Table 3.2. Comparison between the high-order entropy values of the final populations of the introduced EA and ones based on ED and PD. Stat shows the results of a Kruskal-Wallis test at the significance level of 95% with Bonferroni correction. X^+ means the median of the measure is better than the one for variant X, X^- means it is worse and X^* indicates no significant difference.

		$n = 50$								$n = 100$							
		ENT (1)		ED (2)		PD (3)		Range		ENT (1)		ED (2)		PD (3)		Range	
μ	k	**mean**	**stat**	**mean**	**stat**	**mean**	**stat**	H_{min}	H_{max}	**mean**	**stat**	**mean**	**stat**	**mean**	**stat**	H_{min}	H_{max}
12	2	7.09	2^*3^*	7.09	1^*3^*	7.09	1^*2^*	4.6052	7.0901	7.78	2^*3^*	7.78	1^*3^*	7.78	1^*2^*	5.2983	7.7832
12	3	7.09	2^*3^*	7.09	1^*3^*	7.09	1^*2^*	4.6052	7.0901	7.78	2^*3^*	7.78	1^*3^*	7.78	1^*2^*	5.2983	7.7832
12	4	7.09	2^*3^*	7.09	1^*3^*	7.09	1^*2^*	4.6052	7.0901	7.78	2^*3^*	7.78	1^*3^*	7.78	1^*2^*	5.2983	7.7832
20	2	7.60	2^*3^*	7.60	1^*3^*	7.60	1^*2^*	4.6052	7.6006	8.29	2^*3^*	8.29	1^*3^*	8.29	1^*2^*	5.2983	8.2940
20	3	7.60	2^*3^*	7.60	1^*3^*	7.60	1^*2^*	4.6052	7.6006	8.29	2^*3^*	8.29	1^*3^*	8.29	1^*2^*	5.2983	8.2940
20	3	7.60	2^*3^*	7.60	1^*3^*	7.60	1^*2^*	4.6052	7.6006	8.29	2^*3^*	8.29	1^*3^*	8.29	1^*2^*	5.2983	8.2940
50	2	7.80	2^*3^*	7.80	1^*3^*	7.80	1^*2^*	4.6052	7.7997	**9.17**	2^+3^+	9.14	1^-3^*	9.13	1^-3^*	5.2983	9.1965
50	3	**8.52**	2^+3^+	8.51	1^-3^*	8.51	1^-2^*	4.6052	8.5172	9.21	2^*3^*	9.21	1^*3^*	9.21	1^*2^*	5.2983	9.2103
50	4	8.52	2^*3^*	8.52	1^*3^*	8.52	1^*2^*	4.6052	8.5172	9.21	2^*3^*	9.21	1^*3^*	9.21	1^*2^*	5.2983	9.2103
100	2	7.80	2^*3^+	7.80	1^*3^+	7.79	1^-2^-	4.6052	7.8017	9.19	2^+3^+	9.18	1^-3^*	9.16	1^-2^*	5.2983	9.1982
100	3	**9.21**	2^+3^+	9.18	1^-3^*	9.19	1^-2^*	4.6052	9.2103	9.90	2^+3^+	9.90	1^-3^*	9.90	1^-2^*	5.2983	9.9035
100	4	9.21	2^+3^*	9.21	1^-3^-	9.21	1^*2^+	4.6052	9.2103	9.90	2^+3^*	9.90	1^-3^-	9.90	1^*2^+	5.2983	9.9035
500	2	7.80	2^*3^+	7.80	1^*3^+	7.80	1^-2^-	4.6052	7.8036	9.20	2^+3^+	9.20	1^-3^+	9.16	1^-2^-	5.2983	9.1999
500	3	**10.82**	2^+3^+	10.45	1^-3^*	10.60	1^-2^*	4.6052	10.8198	**11.51**	2^+3^+	11.34	1^-3^*	11.45	1^-2^*	5.2983	11.5129
500	4	10.82	2^+3^+	10.76	1^-3^*	10.82	1^-2^*	4.6052	10.8198	11.51	2^+3^+	11.47	1^-3^*	11.51	1^-2^*	5.2983	11.5129
1000	2	7.80	2^*3^+	7.80	1^*3^+	7.79	1^-20^-	4.6052	7.8038	9.20	3^*4^+	9.16	1^*3^+	9.01	1^-2^-	5.2983	9.2001
1000	3	**11.35**	2^+3^+	10.73	1^-3^*	11.03	1^-2^*	4.6052	11.5129	**12.16**	2^+3^+	11.33	1^-3^*	11.89	1^-3^*	5.2983	12.2061
1000	4	**11.52**	2^+3^+	11.30	1^-3^*	11.50	1^-2^*	4.6052	11.5129	**12.21**	3^+4^+	10.36	1^-3^-	11.90	1^-2^+	5.2983	12.2061

3.6.2 Unconstrained Diversity Optimisation

In this section, we first compare classic 2-OPT with biased 2-OPT. Our hypothesis is that biased 2-OPT is more likely to generate offspring that contribute to the population's entropy, while classic 2-OPT may be better at generating tours that satisfy quality constraints. Since no quality constraints are imposed here, we expect that biased 2-OPT will perform better. To compare the two algorithms, we conducted experiments on a complete graph with 100 nodes, using $k = 2$ and $\mu \in 25, 125, 250$.

Figure 3.2 compares the convergence pace of classic 2-OPT and biased 2-OPT. The entropy value is shown on the y-axis, whereby the x-axis represents the number of cost evaluations (iterations). Note that diversity scores shown in the figure are normalised by using Equations 3.1 and 3.2. Figure 3.2 indicates that both operators eventually converge to $H_{\max}$ in most cases. However, biased 2-OPT is faster than the classic 2-OPT, especially with a smaller population.

Figure 3.3 compares the number of cost evaluations required to converge to $H_{\max}$ in the introduced EA using classic and biased 2-OPT over ten runs. One can observe that the number of required cost evaluations is significantly higher for classic 2-OPT. Biased 2-OPT, for example, requires around $2,350$ evaluations on average when $\mu = 25$. On the other hand, the figure is around 14 000 for classic 2-OPT. Moreover, none of the operators converges to $H_{\max}$ within the limit of 100 000 cost evaluations where $\mu = 250$. In this case, the mean of the entropy value of biased and classic 2-OPT are 9.1993 and 9.1983, respectively, while $H_{\max}$ is equal to 9.1994.

Next, we provide a comprehensive comparison between the proposed EA and EAs based on ED and PD proposed by Do et al. [24]. We conduct experiments on all combinations of $n \in \{50, 100\}$, $\mu \in \{12, 20, 50, 100, 500, 1\,000\}$ and $k \in \{2, 3, 4\}$. The termination criteria are reaching either the entropy value of $H_{\max}$ or the limitation of 100 000 cost evaluations. Note that the EAs based on ED and PD compare the offspring to the entire population, requiring considerably more diversity evaluations per generated offspring. Table 3.2 compares the entropy of the final population obtained from the algorithms. Here, we solely use biased 2-OPT due to its efficiency in unconstrained diversity optimisation. The results show that the proposed algorithm outperforms the algorithms based on ED and PD over large populations and long segments (i. e. $\mu \in \{500, 1000\}$ and $k \in \{3, 4\}$). In the case $n = 50$, $k = 3$ and $\mu = 1000$, for instance, the introduced EA scores 11.35 entropy value while the algorithms based on ED and PD achieve 10.73 and 11.03, respectively. According to the pigeon's hole principle, the most challenging cases in unconstrained optimisation are where $2n\mu$ is close to a factor of u (number of segments). For instance, if $n = 100$, $k = 2$, the most challenging cases arise where μ is close to a factor of 49.5. As Table 3.2 shows the EA is incapable of hitting $H_{\max}$ when $n = 100$ and $\mu = 50$. We take a closer look at such cases in Section 3.8.

3.6.3 Constrained Diversity Optimisation

In constrained diversity optimisation, the performance of classic 2-OPT and biased 2-OPT is strongly correlated to the threshold; the wider the threshold, the better the performance of biased 2-OPT, and vice versa. Therefore, we used both operators simultaneously in this subsection. Having selected a parent, we generate two offspring individuals, one by classical 2-OPT, the other by biased version. The offspring that complies with the quality criterion and results in a higher diversity is replaced with the parent. Here, the experiments are conducted on eil51, eil76, and eil101 from the TSPlib, [97] where a threshold of $\alpha = 5\%$ is considered. Moreover, the limit of cost evaluations increases to 300 000 due to the imposition of the quality constraint.

Table 3.3 compares the entropy value of the final population obtained from the introduced algorithm and the algorithms based on ED and PD in [24]. Table 3.3 indicates that the introduced EA outperforms the ones based on ED and PD in most instances. The algorithm based on PD has achieved a better entropy value over only four cases. Given that all these four cases are among the largest ones, a possible reason could be differences in the algorithms' survival selection resulting in slower convergence of the introduced EA than the others in terms of cost evaluations. However, the smaller instances show that if the number of cost evaluations is sufficient, the introduced EA is likely to outperform the others. We conduct another experiment summarised in Figure 3.4 to elaborate more on this matter.

Figure 3.4 illustrates the relationship between the number of cost evaluations and the final population's entropy for the introduced EA on eil101 with $\mu = 500$, $k \in 2, 3, 4$ and $\alpha \in 0.05, 0.1, 0.2$. Similar patterns were observed for the other cases, which are omitted for brevity. The x-axis shows the number of cost evaluations, and the y-axis represents the final population's entropy. The figure reveals that when the number of cost evaluations is low, the introduced EA results in a lower entropy value than the other algorithms. However, it always converges to a higher entropy value. This pattern holds for all nine combinations of k and α. The figure also depicts that as k increases, the introduced EA outperforms the other two algorithms in fewer cost evaluations. When comparing ED and PD, the former converges faster but at a lower entropy level.

Figure 3.5 shows the edges used in the sets of 125 tours obtained from the introduced EA in constrained ($\alpha \in \{0, 0.05, 0.5\}$) and unconstrained diversity optimisation on eil101. The figure clearly demonstrates the proportional relationship between α and population diversity. Additionally, Figure 3.5 compares a population with the entropy value of $H_{\min}$ (left) with a population with $H_{\max}$ entropy (right), highlighting the differences between the two. As the population's entropy increases, the number of incorporated edges (segments) also rises while the frequency of edges decreases.

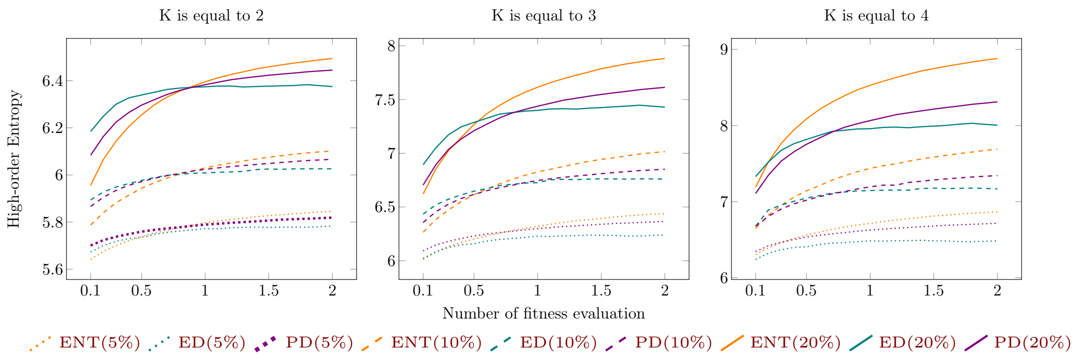


FIGURE 3.4. Impact of number of fitness evaluation, segment size, and threshold on the algorithms in eil101. The percentages shows the allowed threshold.

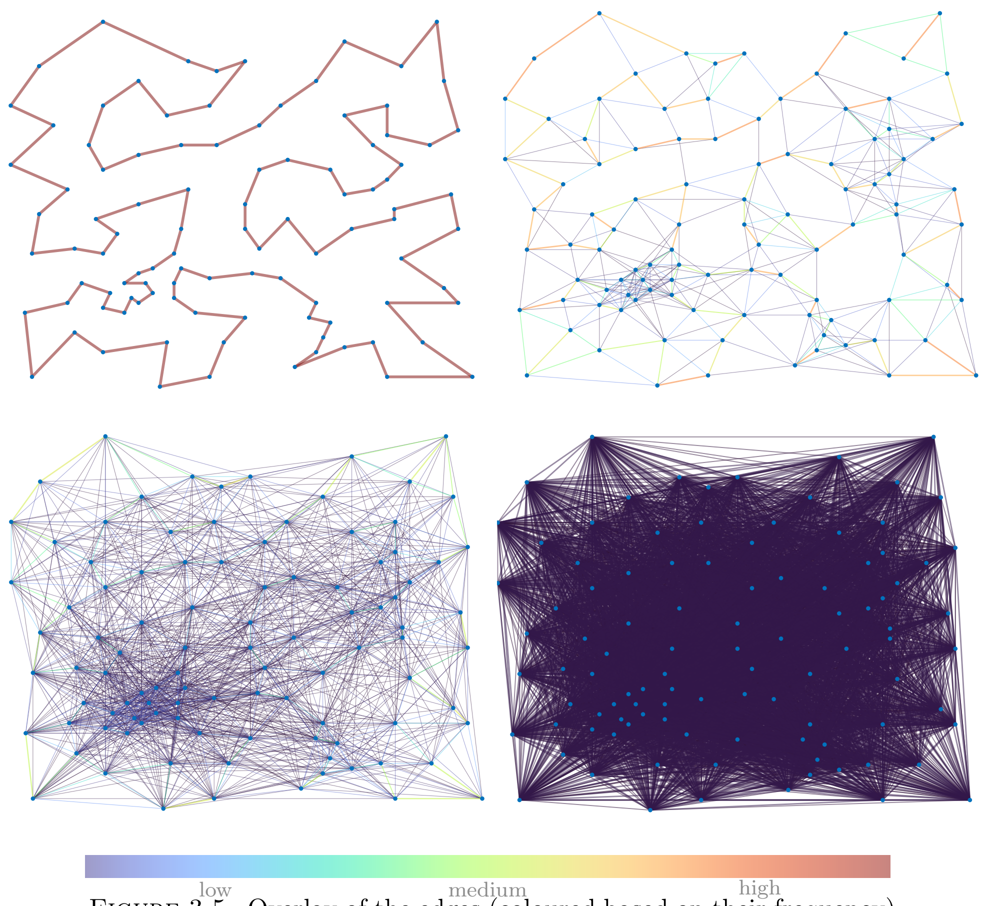


FIGURE 3.5. Overlay of the edges (coloured based on their frequency) incorporated into the population of the introduced EA on eil101 where α increases from 0 to $+\infty$.

TABLE 3.3. Comparison between the high-order entropy values of final populations of the introduced EA and EAs based on ED and PD on TSPlib instances eil51, eil76 and eil101 (threshold is equal to $\alpha = 0.05$). Tests and notations are in line with Table 3.2.

		eil51 ($H_{min} = 4.6250$)						**eil76** ($H_{min} = 5.0239$)						**eil101** ($H_{min} = 5.3083$)					
		ENT (1)		ED (2)		PD (3)		ENT (1)		ED (2)		PD (3)		ENT (1)		ED (2)		PD (3)	
μ	k	**mean**	**stat**	**mean**	**stat**	**mean**	**stat**	**mean**	**stat**	**mean**	**stat**	**mean**	**stat**	**mean**	**stat**	**mean**	**stat**	**mean**	**stat**
12	2	**5.1133**	2^+3^+	5.0586	1^-3^+	5.0381	1^-2^+	**5.4617**	2^+3^+	5.4047	1^-3^*	5.3872	1^-2^*	**5.8137**	2^+3^+	5.7674	1^-3^*	5.7580	1^-2^*
12	3	**5.5648**	2^*3^+	5.4964	1^*3^+	5.4216	1^-2^-	**5.8517**	2^*3^+	5.7699	1^-3^*	5.6977	1^-2^*	**6.2213**	2^*3^+	6.1784	1^-3^+	6.1275	1^-2^-
12	4	**5.7640**	2^+3^+	5.6764	1^*3^*	5.6043	1^-2^*	**6.0499**	2^+3^+	5.9346	1^-3^*	5.8546	1^-2^*	**6.4660**	2^+3^+	6.3742	1^-3^*	6.3058	1^-2^*
20	2	**5.1354**	2^+3^+	5.0543	1^-3^*	5.0687	1^-2^*	**5.4843**	2^+3^+	5.4205	1^-3^*	5.4241	1^-2^*	**5.8232**	2^+3^+	5.7961	1^-3^*	5.7822	1^-2^*
20	3	**5.6557**	2^+3^+	5.4943	1^-3^*	5.5157	1^-2^*	**5.9351**	2^+3^+	5.7911	1^-3^*	6.7810	1^-2^*	**6.3098**	2^+3^+	6.1778	1^-3^*	6.1812	1^-2^*
20	4	**5.9247**	2^+3^+	5.6846	1^-2^*	5.7386	1^-2^*	**6.1831**	2^+3^+	5.9656	1^-3^*	5.9623	1^-2^*	**6.5566**	2^+2^+	6.3810	1^-3^*	6.3834	1^-2^*
50	2	**5.1704**	2^+3^+	5.0618	1^-3^-	5.1017	1^-2^+	**5.5015**	2^+3^+	5.4194	1^-3^-	4.4454	1^-2^+	**5.8262**	2^+3^+	5.7607	1^-3^-	5.7938	1^-2^+
50	3	**5.7371**	2^+3^+	5.5087	1^-3^-	5.6150	1^-2^+	**5.9961**	2^+3^+	5.7861	1^-3^-	5.8497	1^-2^+	**6.3594**	2^+3^+	6.1816	1^-3^-	6.2370	1^-2^+
50	4	**6.0927**	2^+3^+	5.7123	1^-3^-	5.8982	1^-2^+	**6.2776**	2^+3^+	5.9674	1^-3^*	6.0776	1^-2^*	**6.6490**	2^+3^+	6.3997	1^-3^-	6.4858	1^-2^+
100	2	**5.1683**	2^+3^+	5.0623	1^-3^-	5.1033	1^-2^+	**5.4911**	2^+3^+	5.4227	1^-3^-	5.4464	1^-2^+	**5.7980**	2^+3^+	5.7569	1^-3^-	5.7804	1^-2^+
100	3	**5.7503**	2^+3^+	5.5175	1^-2^-	5.6452	1^-2^+	**5.9870**	2^+3^+	5.8120	1^-3^-	5.8658	1^-2^+	**6.2890**	2^+3^+	6.1838	1^-3^-	6.2291	1^-2^+
100	4	**6.1436**	2^+3^+	5.7319	1^-3^-	5.9646	1^-2^+	**6.3027**	2^+3^+	6.0127	1^-3^-	6.1098	1^-2^+	**6.6246**	2^+3^+	6.4137	1^-3^-	6.4938	1^-2^+
500	2	**5.1203**	2^+3^+	5.0396	1^-3^-	5.0815	1^-2^+	**5.4320**	2^+3^*	5.4013	1^-3^-	5.4244	1^*2^+	5.7070	2^*3^-	5.7111	1^*3^-	**5.7377**	1^+2^+
500	3	**5.6794**	2^+3^+	5.5131	1^-3^-	5.6359	1^-2^+	**5.8876**	2^+3^+	5.8077	1^-3^-	5.8653	1^-2^+	6.1379	2^*3^-	6.1180	1^*3^-	**6.1808**	1^+2^+
500	4	**6.0864**	2^+3^+	5.7660	1^-3^-	5.9991	1^-2^+	**6.2218**	2^+3^+	6.0399	1^-3^-	6.1469	1^-2^+	**6.5770**	2^+3^*	6.3648	1^-3^-	6.4616	1^*2^+
1000	2	**5.0909**	2^+3^+	5.0187	1^-3^-	5.0585	1^-2^+	5.4074	2^+3^*	5.3811	1^-3^-	**5.6926**	1^*2^+	5.7194	2^*3^-	5.6933	1^*3^*	5.7194	1^+2^*
1000	3	**5.6291**	2^+3^+	5.4760	1^-3^-	5.5943	1^*2^+	**5.8442**	2^+3^*	5.7712	1^-3^-	5.8333	1^*2^+	6.0987	2^*3^-	6.0891	1^*3^-	**6.1495**	1^*2^+
1000	4	**6.0238**	2^+3^+	5.7357	1^-3^-	5.9498	1^*2^+	**6.1771**	2^+3^+	6.0039	1^-3^-	6.1116	1^-2^+	**6.4372**	2^+3^*	6.3372	1^-3^-	6.4348	1^*2^+

3.7 Evolutionary Diversity Optimisation with a $(\mu+\lambda)$ EA

Most studies in the EDO literature use a $(\mu + 1)$ EA to maximise the diversity of tours, where the offspring can replace either the parent or one of the individuals in the population P (see, e.g., [12] or [76]). In the preceding section, exactly two offspring individuals are generated in every iteration: one by 2-OPT and the other by biased 2-OPT. However, similar to the referenced studies, only one offspring can survive to the next generation to avoid increasing the complexity of the problem. Unlike classical optimisation problems, where the fitness of a solution is independent of other solutions, diversity optimisation problems require the calculation of diversity values for a set of solutions, making the fitness of solutions dependent on other individuals in the population. This increases the complexity of the problem by adding a subset selection sub-problem: how do we select a subset of μ individuals from a set of $\mu + \lambda$ individuals efficiently? Nevertheless, increasing the size λ of the children's pool by using a $(\mu + \lambda)$ EA can potentially boost the performance of the EA if a suitable method for subset selection is incorporated into the algorithm.

This section aims to answer two questions: 1) Is it beneficial to increase the pool of offspring? 2) Given the complexity of the diversity problem, what is the best survival selection approach to determine the next generation?

Algorithm 6 Diversity maximising $(\mu + \lambda)$ EA

```
Require: Population size $\mu$, size of offspring pool $\lambda \geq 1$
1: Initialise the population $P$ with $\mu$ TSP tours such that
   $c(I) \leq (1 + \alpha) \cdot OPT$ for all $I \in P$.
2: Set $Q = \emptyset$.
3: while $|Q| < \lambda$ do
4:     Choose $I \in P$ uniformly at random and produce an offspring $I'$ of $I$ by mutation.
5:     If $c(I') \leq (1 + \alpha) \cdot OPT$, add $I'$ to $Q$.
6: Set $P = \texttt{SELECT}(S = P \cup Q)$.                {Subset selection.}
7: Repeat steps 2 to 6 until a termination criterion is reached.
```

Recall that in Algorithm 5 in every iteration, exactly two offspring individuals are generated, one by 2-OPT and the other by biased 2-OPT. A greedy survival selection strategy then decides which individual survives (only one of the offspring individuals can survive to the next generation). Algorithm 6 implements a generalisation of the $(\mu + 1)$ EA in Algorithm 5) with a parameterisable subset selection procedure (see call to function `SELECT` in line 6 of Algorithm 6). In order to focus solely on the effect of subset selection themes, offspring individuals are generated by unbiased mutation only. The main challenge is, given a population P with $|P| = \mu$, a set of offspring individuals Q with $|Q| = \lambda \geq 1$, to select a subset S^* of the union set $S = P \cup Q$ such that

$$S^* = \underset{S' \subset S, |S'| = \lambda}{\arg\max} \; H(S \setminus S').$$

There are $\binom{\mu+\lambda}{\mu}$ possible subsets. As a consequence, iterating over all these subsets is computationally infeasible. We therefore tackle this subset-selection problem by means of three different heuristics.

Algorithm 7 Greedy subset selection.

```
Require: Multi-set $S$ with $|S| = \mu + \lambda$
1: while $|S| > \mu$ do
2:     Remove one individual $p$ from $S$, where $p = \arg\max_{q \in S} H(S \setminus \{q\})$.
```

Greedy selection

The greedy selection strategy (see Algorithm 7) is a straightforward generalisation of the greedy survival selection adopted in Algorithm 5. In the ith iteration, the algorithm removes an individual from the set S whose deletion leads to a locally optimal maximisation of the diversity measure. Note that this selection method is computationally costly. It requires, assuming that all λ offspring individuals adhere to the quality constraint (see line 5 in Algorithm 6), $(\mu + \lambda - i)$ H-evaluations in the ith iteration for $i = 0, \ldots, \lambda - 1$ and thus the overall number of H-evaluations is

$$\sum_{i=0}^{\lambda-1}(\mu + \lambda - i) = \sum_{i=1}^{\lambda}(\mu + i) = \lambda\mu + \frac{\lambda(\lambda+1)}{2} = \lambda\left(\frac{2\mu + \lambda + 1}{2}\right) = \Theta(\lambda\mu + \lambda^2).$$

Algorithm 8 Tournament subset selection.

```
Require: Multi-set $S$ with $|S| = \mu + \lambda$, tournament size $r \geq 2$
1: while $|S| > \mu$ do
2:     Select a subset $R \subseteq S$ of $r$ individuals uniformly at random.
3:     Remove one individual $p$ from $S$, where $p = \arg\max_{q \in R} H(S \setminus \{q\})$.
```

Tournament selection

We adopt a tournament selection strategy iteratively (see Algorithm 8). In λ iterations, the algorithm samples a subset R with $|R| = r$ – $r \geq 2$ being the tournament size – individuals from the set uniformly at random (with replacement). Eventually, the individual $q \in R$ is dropped, whose deletion yields the maximal H-value. The number of H-evaluations is $\lambda \cdot r$ since the H-value is calculated r times per iteration, and it takes λ iterations until the set is reduced to μ individuals. Note that this is $\Theta(\lambda)$ for constant r.

EA-based selection

The subset-selection process itself is optimised with a $(1 + 1)$ EA. To this end we maintain a bitstring $x \in \{0, 1\}^{\mu+\lambda}$ and optimise the diversity of the set $S' = \{p_i \in S \mid x_i = 1\}$ under the equality constraint $|x|_1 = \mu$ (see Algorithm 9). The algorithm is initialised such that exactly μ individuals are selected (i.e., $|x|_1 = \mu$). In L iterations – L being a parameter – the algorithm generates an offspring y by flipping zero-bits to

Algorithm 9 $(\mu + \lambda)$ EA-based subset selection.

```
Require: Multi-set $S$ with $|S| = \mu + \lambda$
1: Initialise $x \in \{0,1\}^{|S|}$ where $|x|_1 = \mu$.
2: for $L$ iterations do
3:    Generate $y$ from $x$ by flipping one-bits independently with probability of $w/\mu$, where $w$ is selected based on power of law distribution. Then flip the same number of bits flip from zero to one randomly to maintain the equality constraint, i. e., $|y|_1 = |x|_1 = \mu$).
4:    if $F(y) \geq F(x)$ then {Refer to Equation 3.17}
5:       Replace $x$ with $y$.
```

ones and the same amount of one-bits to zero in the parent x. The number of flipped bits is $2 \cdot l$ where l is sampled from a Binomial distribution with parameters $\mu + \lambda$ and $\frac{w}{n} \in [0, 1]$ where w is selected based on a power-law distribution as it is done in the heavy-tailed mutation [29]. This is done to allow the algorithm to escape from being trapped in local optima by increasing the probability of flipping multiple one and zero bits. In addition, we introduce a diversity mechanism based on the length of the tour to avoid getting stuck in local optima. We first calculate H_0 and l_0, which are the diversity of the population P and the summation of the length of individuals in P, respectively. The fitness of offspring in Algorithm 9 can be calculated from:

$$F(x) = \begin{cases} \frac{\Delta H}{\Delta l} & \text{if } \Delta H > 0 \text{ and } \Delta l \geq 0 \\ M & \text{if } \Delta H \geq 0 \text{ and } \Delta l < 0 \\ -M & \text{if } \Delta H < 0 \end{cases} \tag{3.17}$$

where ΔH and Δl are $H(y) - H_0$ and $l(y) - l_0$, respectively. We use F as the fitness function, a higher F, a fitter solution. Note that L is set to $2\mu\lambda$ in this study based on preliminary investigations. Note that if $\Delta H \geq 0$ and $\Delta l < 0$, we stop the search for the iteration and go to the next generation.

3.8 Experimental Investigation

Analogous to the experiments in Section 3.6, we conduct two series of empirical investigations to study the impact of an offspring pool $\lambda \geq 2$. We first look at the algorithms' performance in the setting of unconstrained diversity optimisation. Then, we analyse constrained diversity optimisation. For the sake of fair comparison, we consider the number of diversity evaluations (H-evaluations) as the termination criterion. Moreover, we consider $k = 2$ in these series of experiments.

3.8.1 Unconstrained Diversity Optimisation

We first scrutinise the algorithms' performance in the unconstrained case, where no constraints are imposed on the quality of the tours. We conduct the experiments on a complete graph with $n = 50$ nodes, and consider the following values for the other

TABLE 3.4. Comparison of the proposed subset selection schemes and $(\mu+1)$ EA in the unconstrained optimisation setting. The column *rate* shows the number of times obtaining the optimal diversity out of 30 independent runs.

	$(\mu+\lambda)$ **EA**			**Tournament**			**Greedy**			$(\mu+1)$ **EA**		
μ	H	Stat (1)	rate	H	Stat (2)	rate	H	Stat (3)	rate	H	Stat (4)	rate
20	7.6	$2^+3^+4^*$	**30**	6.83	$1^-3^-4^-$	0	7.49	$1^-2^+4^-$	0	7.6	$1^*2^+3^+$	29
21	7.65	$2^+3^+4^*$	**30**	6.86	$1^-3^-4^-$	0	7.53	$1^-2^+4^-$	0	7.65	$1^*2^+3^+$	18
22	7.7	$2^+3^+4^*$	**30**	6.89	$1^-3^-4^-$	0	7.57	$1^-2^+4^-$	0	7.69	$1^*2^+3^+$	11
23	7.74	$2^+3^+4^+$	**30**	6.92	$1^-3^-4^-$	0	7.6	$1^-2^+4^-$	0	7.73	$1^-2^+3^+$	0
24	7.78	$2^+3^+4^+$	**10**	6.93	$1^-3^-4^-$	0	7.63	$1^-2^+4^-$	0	7.77	$1^-2^+3^+$	0
25	7.8	$2^+3^+4^+$	**16**	6.96	$1^-3^-4^-$	0	7.65	$1^-2^+4^-$	0	7.78	$1^-2^+3^+$	0
26	7.78	$2^+3^+4^+$	**30**	6.98	$1^-3^-4^-$	0	7.67	$1^-2^+4^-$	0	7.78	$1^-2^+3^+$	1
27	7.77	$2^+3^+4^*$	**30**	7	$1^-3^-4^-$	0	7.69	$1^-2^+4^-$	0	7.77	$1^*2^+3^+$	24
28	7.76	$2^+3^+4^*$	**30**	7.03	$1^-3^-4^-$	0	7.7	$1^-2^+4^-$	0	7.76	$1^*2^+3^+$	29
29	7.76	$2^+3^+4^*$	**30**	7.04	$1^-3^-4^-$	0	7.71	$1^-2^+4^-$	0	7.76	$1^*2^+3^+$	**30**
30	7.75	$2^+3^+4^*$	**30**	7.07	$1^-3^-4^-$	0	7.71	$1^-2^+4^-$	0	7.75	$1^*2^+3^+$	**30**

parameters: $k = 2$, $\lambda = 12$, and $\mu \in \{20, 21, \ldots, 30\}$. The termination criterion is set to 5×10^7 diversity evaluations. According to the pigeonhole principle, the most challenging case in unconstrained diversity optimisation occurs when $2n\mu$ is a factor of u (the number of segments), as observed in Table 3.2. While Algorithm 9 can achieve $H_{\max}$ in cases where $2n\mu$ is not close to any factor of u, it cannot do so when $n = 50$, $k = 2$, and $\mu = 25$. For $n = 50$ and $k = 2$, the most difficult cases arise when μ is any factor of 24. Thus, we set $\mu \in \{20, 21, \ldots, 30\}$ to take such challenging cases into account. Table 3.4 summarises the results of these experiments. One can observe that the EA-based subset selection algorithm 9 outperforms the competitors, followed by the conventional $(\mu+1)$ EA and the Greedy algorithm. The EA algorithm always hits $H_{\max}$ on all instances except where $\mu \in \{24, 25\}$. In these cases, the EA selection hits the optimum 10 and 16 times, respectively. The $(\mu + 1)$ EA cannot bring about the optimal population when $\mu \in \{23, 24, 25\}$ out of 30 runs, and it hits the optimum only one time when $\mu = 26$. The $(\mu + 1)$ EA performance gets better as we deviate from $\mu = 24$, and it results in the optimum for the two largest μ values in all 30 independent runs, similar to the EA algorithm. Moreover, the statistical observation confirms a significant difference between the EA algorithm and the $(\mu + 1)$ EA in cases $\mu \in \{23, \ldots, 25\}$. The tournament and the greedy algorithm never resulted in the optimal values in this setting. In fact, the tournament algorithm's results are far away from the optimums.

In conclusion, we can confirm that an offspring pool of $\lambda \geq 2$ can boost the algorithm performance in the unconstrained optimisation if a suitable subset selection method such as the EA-based approach in 5 is employed. Next, we analyse the methods where a quality constraint is imposed on the problem.

TABLE 3.5. Comparison of the proposed subset selection schemes and $(\mu+1)$ EA. The termination criterion is 10^7 H-evaluations.

			EA		**Tournament**		**Greedy**		$(\mu+1)$ **EA**	
Inst.	α	μ	H	Stat (1)	H	Stat (2)	H	Stat (3)	H	Stat (4)
st70	0.05	50	5.67	$2^{-}3^{+}4^{+}$	**5.7**	$1^{+}3^{+}4^{+}$	5.64	$1^{-}2^{-}4^{+}$	5.62	$1^{-}2^{-}3^{-}$
st70	0.05	100	5.7	$2^{*}3^{+}4^{+}$	**5.73**	$1^{*}3^{+}4^{+}$	5.65	$1^{-}2^{-}4^{*}$	5.65	$1^{-}2^{-}3^{*}$
st70	0.12	50	**5.98**	$2^{+}3^{*}4^{+}$	5.95	$1^{-}3^{-}4^{+}$	5.97	$1^{*}2^{+}4^{+}$	5.93	$1^{-}2^{-}3^{-}$
st70	0.12	100	**6.01**	$2^{+}3^{+}4^{+}$	5.98	$1^{-}3^{*}4^{+}$	5.98	$1^{-}2^{*}4^{+}$	5.96	$1^{-}2^{-}3^{-}$
st70	0.25	50	**6.28**	$2^{+}3^{*}4^{+}$	6.2	$1^{-}3^{-}4^{-}$	**6.28**	$1^{*}2^{+}4^{+}$	6.25	$1^{-}2^{+}3^{-}$
st70	0.25	100	**6.3**	$2^{+}3^{*}4^{+}$	6.25	$1^{-}3^{-}4^{-}$	**6.3**	$1^{*}2^{+}4^{+}$	6.28	$1^{-}2^{+}3^{-}$
eil101	0.05	50	6.02	$2^{-}3^{+}4^{+}$	**6.04**	$1^{+}3^{+}4^{+}$	5.97	$1^{-}2^{-}4^{+}$	5.93	$1^{-}2^{-}3^{-}$
eil101	0.05	100	6.05	$2^{-}3^{+}4^{+}$	**6.06**	$1^{+}3^{+}4^{+}$	5.98	$1^{-}2^{-}4^{*}$	5.97	$1^{-}2^{-}3^{*}$
eil101	0.12	50	**6.35**	$2^{+}3^{+}4^{+}$	6.31	$1^{-}3^{-}4^{*}$	6.34	$1^{-}2^{+}4^{+}$	6.3	$1^{-}2^{*}3^{-}$
eil101	0.12	100	**6.37**	$2^{+}3^{+}4^{+}$	6.34	$1^{-}3^{*}4^{+}$	6.34	$1^{-}2^{*}4^{+}$	6.33	$1^{-}2^{-}3^{-}$
eil101	0.25	50	**6.73**	$2^{+}3^{*}4^{+}$	6.63	$1^{-}3^{-}4^{-}$	**6.73**	$1^{*}2^{+}4^{+}$	6.7	$1^{-}2^{+}3^{-}$
eil101	0.25	100	**6.75**	$2^{+}3^{+}4^{+}$	6.68	$1^{-}3^{-}4^{-}$	6.74	$1^{-}2^{+}4^{+}$	6.73	$1^{-}2^{+}3^{-}$
a280	0.05	50	6.87	$2^{-}3^{+}4^{+}$	**6.88**	$1^{+}3^{+}4^{+}$	6.84	$1^{-}2^{-}4^{+}$	6.82	$1^{-}2^{-}3^{-}$
a280	0.05	100	6.89	$2^{-}3^{+}4^{+}$	**6.9**	$1^{+}3^{+}4^{+}$	6.85	$1^{-}2^{-}4^{*}$	6.84	$1^{-}2^{-}3^{*}$
a280	0.12	50	**7.15**	$2^{+}3^{+}4^{+}$	7.11	$1^{-}3^{-}4^{+}$	7.13	$1^{-}2^{+}4^{+}$	7.1	$1^{-}2^{-}3^{-}$
a280	0.12	100	**7.16**	$2^{+}3^{+}4^{+}$	7.15	$1^{-}3^{*}4^{+}$	7.14	$1^{-}2^{*}4^{+}$	7.14	$1^{-}2^{-}3^{-}$
a280	0.25	50	**7.49**	$2^{+}3^{+}4^{+}$	7.4	$1^{-}3^{-}4^{-}$	7.48	$1^{-}2^{+}4^{+}$	7.45	$1^{-}2^{+}3^{-}$
a280	0.25	100	7.45	$2^{*}3^{-}4^{-}$	7.45	$1^{*}3^{-}4^{-}$	**7.49**	$1^{+}2^{+}4^{*}$	7.48	$1^{+}2^{+}3^{*}$

3.8.2 Constrained Diversity Optimisation

We conduct a series of experiments to evaluate the proposed methods' performance in constrained diversity optimisation. We test the algorithms on the instances st70, eil101, and a 280 from the TSPlib [97]. We set the tournament size to $r = 3$, the offspring pool size to $\lambda = 50$, and experiments with several values for the quality constraint $\alpha = \{ 0.05, 0.12, 0, 25\}$ and the population size $\mu = \{50, 100\}$. The termination criterion is set to 10^7 H-evaluations.

The results are summarised in Table 3.5. The table presented in this section reveals some interesting findings. Firstly, the tournament selection method outperforms the other methods when the smallest considered α-value ($\alpha = 0.05$) is used, with the EA algorithm following closely behind. Conversely, the vanilla $(\mu + 1)$ EA performs the worst in these cases. This may be due to its inferior exploration capabilities, as K-tournament selection encourages exploration over exploitation. As α increases to 0.12, the EA-based selection method produces the best results, followed by the greedy and tournament methods. The greedy algorithm produces sets with slightly higher diversity when compared to tournament selection, but again, the $(\mu+1)$ EA performs poorly with respect to diversity. For $\alpha = 0.25$, the greedy algorithm performs best for $\mu = 50$, while the EA outperforms the others for $\mu = 100$, with the $(\mu + 1)$ EA ranking closely behind. The tournament selection method performs worst in these cases. Overall, the EA exhibits the most stable performance across all instances,

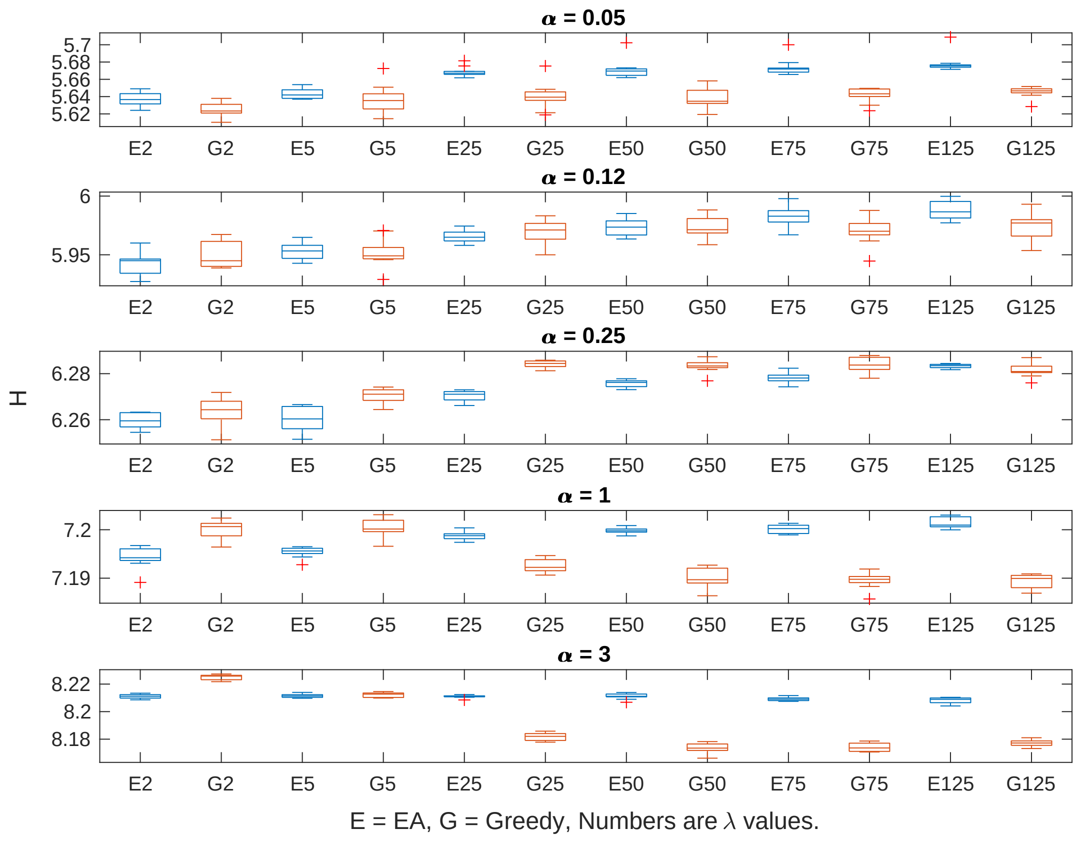


FIGURE 3.6. Distributions of H-values of the final populations based on 10 independent runs for $(\mu + \lambda)$ EA (E) and Greedy (G) subset selection on instance st70. The plots show results for $\alpha \in \{0.02, 0.12, 0.25, 1, 3\}$ (row-wise) and $\lambda \in \{2, 5, 50, 125\}$ (from left to right).

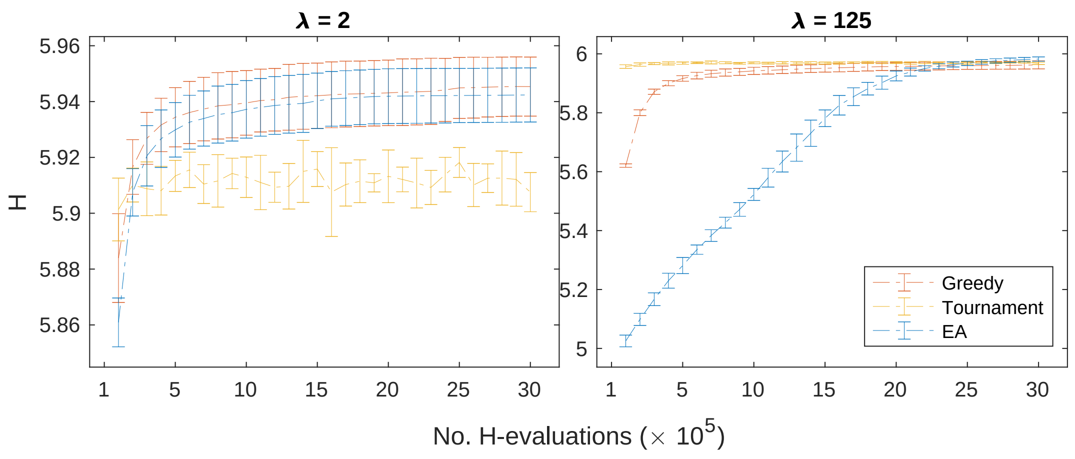


FIGURE 3.7. Representative trajectories of the proposed subset selection methods on instance st70 with $\alpha = 0.12$ and $\lambda \in \{2, 125\}$.

always ranking as the best or second-best method. This may be attributed to the balance between exploration and exploitation that the EA offers. Selection methods with more exploration appear to perform better with tight quality constraints, whereas algorithms that require more exploitation perform better when α is large. Finally, we examine the impact of different values of λ on the performance of the EA and greedy selection methods.

Figure 3.6 depicts the performance of the EA and greedy methods on st70 when $\alpha \in \{0.02, 0.12, 0.25, 1, 3\}$ and $\lambda \in \{2, 5, 50, 125\}$. Figure 3.6 shows that the performance of the EA is slightly better for larger λ-values $(75, 125)$, while it strongly depends on the α-value for the greedy approach. The greedy method preforms slightly better with large λ when α is small (0.05), it is the other way around in the cases of large α-values $(\alpha = 3)$. As $\lambda \to 1$, the greedy algorithm becomes equivalent to the vanilla $(\mu + 1)$ EA. We observed that $(\mu + 1)$ EA outperforms the greedy method in unconstrained diversity. Thus, one could guess that in cases where the quality constraint is relaxed, smaller values of λ work better for the greedy method. In general, the EA's performance is more stable and gets less affected by different values of α, μ, and λ. Figure 3.7 illustrates representative trajectories of the proposed selection methods on st70 where $\alpha = 0.12$ and $\lambda = \{2, 125\}$. The tournament selection has the fastest convergence, and the increase in λ affects the convergence pace less compared to the other methods. The increase in λ has the most significant impact on the EA's convergence pace since there is a linear relation between L and λ (we set $L = 2\mu\lambda$). Therefore, when λ increases, more H-evaluations are spent on each loop of the EA selection. The increase in λ also makes an impact on the greedy convergence pace, but not as severely as it does on the EA selection.

3.9 Conclusion

In the context of EDO, we introduced a method to evolve diverse sets of solutions meeting minimal quality criteria. We adopted a new diversity measure based on high-order entropy to maximise the diversity of a population of TSP solutions. The diversity measure allows equalising the share of segments of multiple nodes, whereas previously proposed diversity measures by [24] in the TSP context focus on the frequency of single edges in the population. We show theoretical properties that a maximally/minimally diverse set of solutions has to fulfil. Furthermore, we study the effects of the high-order entropy measure embedded into a simple population-based evolutionary algorithm experimentally. This algorithm uses different versions of 2-OPT mutations partially biased towards favouring high-frequency segments in TSP tours. Our results in the unconstrained setting without quality restriction and the constrained setting on TSPlib instances show the superiority of the proposed approach if the number of cost evaluations is high. Moreover, we introduced three subset selection methods that enable us to replace more than one individual with offspring for the next generation. Due to the dependency of individuals in the diversity calculation, most studies used the standard $(\mu + 1)$ EA, where at most one individual is replaced in each generation. Our results show that an EA-based selection method as a sub-procedure of a $(\mu + \lambda)$ EA with $\lambda \geq 2$ outperforms the standard $(\mu + 1)$ EA and the other proposed selection methods in most cases.

For future studies, there are several real-world optimisation problems in which EDO can be beneficial and provide practitioners with multiple high-quality solutions. Also, EDO is yet to be studied in multi-objective optimisation problems.

Chapter 4

EAX-based Crossover in the Traveling Salesperson Problem

In the previous chapter, we introduced an EDO-based algorithm specifically designed for computing a structurally diverse set of TSP tours. In line with most studies in the literature on EDO, we made the assumption that the optimal solution is known a priori, as discussed in Chapter 3. In this chapter, we extend our investigation to consider scenarios where the optimal solution may be unknown. Additionally, we adopt EAX [72] in the context of evolutionary diversity optimisation and introduce an approach called EAX-EDO in order to obtain high-quality tours for the TSP while maximising the diversity of the population simultaneously.

In classical EAs, typically, a loss of diversity in the population can be observed when improving the quality of solutions. Although EAX benefits from an entropy-based diversity preservation mechanism, it merely focuses on avoiding premature convergence rather than maximising the diversity of the final population. To address this limitation, we adopt the state-of-the-art EAX crossover operator for solving TSP instances and introduce a modification called EAX-EDO crossover, which is tailored towards simultaneous optimisation of solution quality and population diversity.

We incorporate the EAX-EDO crossover into three EDO approaches. In the case of unknown optimal solutions, we introduce two algorithms. First, we adopt a two-stage framework from the literature of EDO, which alternates between phases of optimising the cost and optimising the diversity (the two-stage EAX-EDO). Second, we introduce the single-stage EAX-EDO designed to simultaneously optimise both the quality of solutions and population diversity. Moreover, we conduct a comparison between the frameworks with the classical EAX and the exact Gurobi optimiser.

Our experimental investigations show that EAX-EDO is capable of generating solutions with very decent quality. More importantly, EAX-EDO can maintain and even increase the diversity of the population during the process of optimising the quality of solutions. This is while EAX requires the sacrifice of the diversity of the

Algorithm 10 EAX-EDO Crossover

```
Require: Two parent tours $p_1$ and $p_2$.
1: Derive an AB-cycle from $p_1$ and $p_2$
2: Construct an intermediate tour from $p_1$ by removing the edges of $E(p_1)$ and adding edges of $E(p_2)$ in the AB-cycle.
3: Count the number of sub-tours in the intermediate solution and store it in $n_{sub}$.
4: while $n_{sub} > 2$ do
5:     Connect two sub-tours based on the neighbourhood search A and update $n_{sub}$.
6: if $n_{sub} = 2$ then
7:     Connect two sub-tours based on the neighbourhood search B.
```

population to gain solutions of higher quality. Moreover, we conduct a series of experiments to examine the robustness of the four competitors' populations against minor changes. The outcome indicates the single-EAX-EDO's superiority in terms of populations' robustness.

The work of this chapter is based on a conference paper [82] presented at the foundations of genetic algorithms (FOGA 2021). The remainder of the chapter is structured as follows. We present the EAX-EDO crossover in Section 4.1 and evolutionary algorithms for the case where an optimal solution is known (Section 4.2) or unknown (Section 4.3). In Section 4.4, we conduct a series of experiments showing the advantage of EAX-EDO when the optimal solution for the TSP is known and unknown on TSP benchmark instances. Finally, we finish with some concluding remarks.

4.1 EAX-EDO Crossover

EAX crossover (EAX CO) is a permutation-based crossover operator which is used in a genetic algorithm (GA) named EAX as well. It was first introduced by Nagata [69]. Several versions of EAX can be found in the literature [70–72]. The version introduced by Nagata and Kobayashi [72] constitutes a key component in one of the best-performing incomplete algorithms for solving the TSP. It is shown that the algorithm is capable of obtaining the optimal or best-known solutions for most benchmark instances and, more importantly, improved 11 best-known solutions. The GA works in two stages. The first stage, which is the main part of the algorithm uses EAX-1AB for crossover, while EAX-Block2 crossover is utilised in the second stage. While EAX-1AB results in offspring very similar to one of the parents, EAX-Block2 brings about more different offspring. Having the first stage converged and incapable of improving the objective function any further, the second stage is initialised to explore solution areas for possible improvements. The algorithm also benefits from an entropy-based diversity mechanism.

During each iteration, the GA generates λ tours from the same parents. Among those tours, it chooses the tours that improve the cost of the solution compared to its parents as well as the entropy score of the population simultaneously if there is such a tour; otherwise, it selects the tour with minimum loss of entropy per improvement of the cost. It is crucial to notice that entropy is used merely to avoid premature convergence and not for the sake of a structurally diverse final population. We refer the interested reader to [72] for more details about the variants of EAX and the diversity preservation mechanism.

In this chapter, we utilise EAX-1AB due to the efficiency and simplicity of this version compared to other variants. EAX-1AB consists of three steps. Let p_1 and p_2 be two parents selected from the population, and $E(p_1)$ and $E(p_2)$ be the sets of edges forming p_1 and p_2, respectively. Firstly, an AB-cycle is derived from p_1 and p_2. An AB-cycle (see Figure 4.1.2) is a cycle where edges of $E(p_1)$ and $E(p_2)$ are linked, alternately. To form an AB-cycle, we start from a random node v. Then, we randomly select an edge (v, u) from $E(p_1)$ and set $v = u$. In a similar manner, we add another edge to the tracing path from p_2 going through v, and reset v. We continue tracing nodes between p_1 and p_2 until an AB-cycle is formed in the trace path. Next, an intermediate solution t (see Figure 4.1.3) is constructed from p_1 by adding edges of $E(p_2)$ and removing edges of $E(p_1)$ in the AB-cycle. Finally, a tour is generated by connecting all sub-tours of the intermediate solution (see Figure 4.1.4). Note that some AB-cycles are formed by two overlapping edges, one from P_1, the other from p_2. Such an AB-cycle is ineffective because it results in an intermediate solution, same as p_1. In this case, discard the ineffective AB-cycle from the tracing path, set v to the last node in the tracing path, and this time select the other edge going through v (there are always two edges going through v).

Algorithm 10 outlines EAX-EDO Crossover (EAX-EDO CO). The difference between EAX-EDO CO and EAX-1AB is the last step, where the sub-tours are connected into a valid TSP tour (compare Figure 4.1.4.a (The Above) and Figure 4.1.4.b). In EAX, the sub-tour r with the minimum number of edges is selected and connected to another sub-tour r' by removing an edge from each of them and adding two new edges. For this purpose, 4-tuples of edges are selected such that $\{e_1, e_2, e_3, e_4\} = \arg\max\{-d(e_1) - d(e_2) + d(e_3) + d(e_4)\}$ where $e_1 \in E(r)$ and $e_2 \in E(t) \setminus E(r)$. Where $E(r)$ and $E(t)$ denotes the set of edges formed sub-tour r and the intermediate solution t. For the sake of reduced computational cost, the search is limited in the way that either end of e_3 should be among the N_{near} nearest nodes to either end of e_3 (here, N_{near} is set to 10). we refer these steps as neighborhood search A. In EAX-EDO CO, the neighbourhood search A is implemented until two sub-tours are left. Then, neighbourhood search B is started. First, all possible 4-tuples of edges complying $c(r) + c(r') - d(e_1) - d(e_2) + d(e_3) + d(e_4) \leq c_{max}$ are stored. From all the possible candidates, the 4-tuple of edges is selected where $\{e_1, e_2, e_3, e_4\} = \arg\max\{\Delta h(e_1) + \Delta h(e_2) + \Delta h(e_3) + \Delta h(e_4)\}$. Here, $\Delta h(e_i)$ is the difference to the contribution of edge i when it is either added or removed from the intermediate solution.

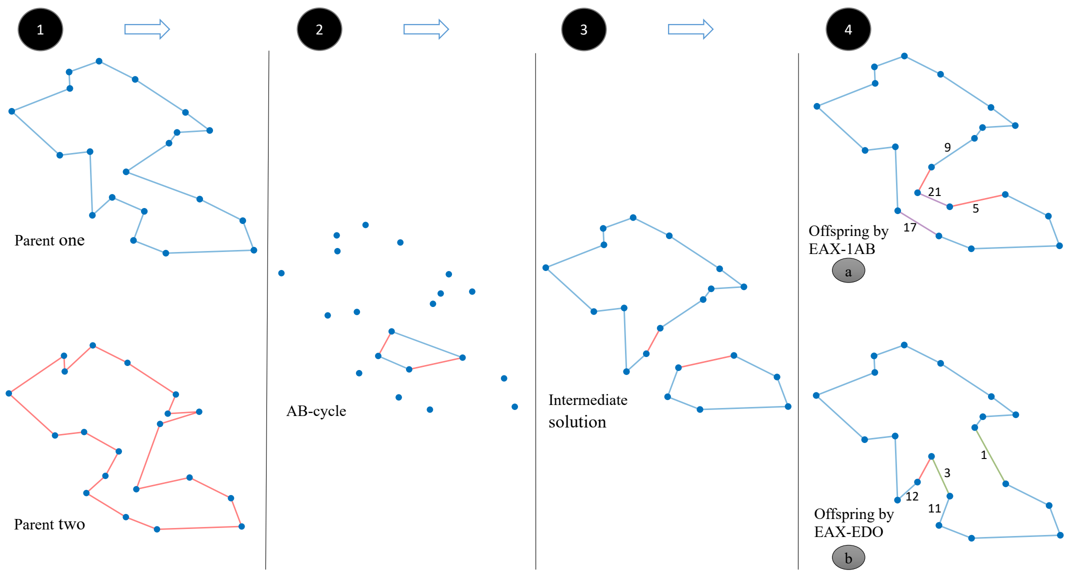


FIGURE 4.1. The representation of the steps to implement EAX-1AB and EAX-EDO. Up to step three the process of implementation for both crossover operators is the same. In step four, EAX-1AB (top) constructs the shortest tour possible, while EAX-EDO (bottom) generates a tour contributing to entropy the most while employing quality threshold (the numbers correspond to occurrences of associated edges in an imaginary population).

4.2 EAX-EDO for Known Optimal Solution

A wide range of highly successful algorithms have been proposed in the literature to solve the TSP, and optimal solutions can be obtained for a wide range of even large instances. In the following, we introduce an EAX-EDO approach that starts with an optimal solution and computes a diverse set of high-quality solutions based on it. The algorithm is outlined in Algorithm 11. The algorithm is initialised with μ copies of an optimal tour. Then, the entropy value of the population is calculated and stored in H. In this stage, H should be equal to H_{min}. Having selected the parents, one offspring is generated by EAX-EDO CO. If the cost of the offspring is at most $c_{max} = (1+\alpha)OPT$, it is added to P; then, an individual with $\arg\max H(P \setminus \{p\})$ is removed. Otherwise, the offspring is discarded. We repeat these steps until a termination criterion is met. Note that since the algorithm is initialized with μ copies of a single tour, a mutation operator should be used in initial iterations (1 000 fitness evaluation in our experiments). This is because EAX-EDO, as a crossover operator, requires two different parent solutions to generate a new offspring. For this purpose, we used 2-OPT in this study. 2-OPT is a random neighbourhood search. In 2-OPT, offspring is first formed by coping all the parent's edges. Then, having removed two edges randomly, we connect each end of the edges to the other edge in a way that it forms a complete TSP tour.

Algorithm 11 Diversity-Maximising-EA

```
Require: Population $P$, minimal quality threshold $c_{max}$
1: while termination criterion is not met do
2:     Choose $p_1$ and $p_2 \in P$ based on parent selection procedure, produce one offspring $p_3$ by crossover.
3:     if $c(p_3) \leq c_{max}$ then
4:         Add $p_3$ to $P$.
5:     if $|P| = \mu + 1$ then
6:         Remove one individual $p$ from $P$, where $p = \arg\max_{q \in P} H(P \setminus \{q\})$.
```

4.3 EAX-EDO for Unknown Optimal Solution

We now discuss the more general case where an optimal solution is unknown. In this case, it is important to diversify the set of solutions and increase the quality of the solution set. Ulrich and Thiele [112] introduced an approach to achieve both goals. In this approach, the algorithm switches between cost minimisation and diversity maximisation. The approach is shown in Algorithm 12. It starts with a cost minimisation phase (see Algorithm 13). The cost minimisation algorithm is initialised with a population optimised by 2-OPT in terms of cost (as it is done by the EAX genetic algorithm in [72]). Next, two individuals are selected to serve as the parents and one tour is generated by EAX-1AB CO. The offspring replaces the first parent if it has a lower cost. Otherwise, it is discarded. These steps continue until an inner termination criterion is met for the cost optimisation stage. The worst found solution within the population dictates the least quality threshold c_{max} for the diversity optimisation phase (see Algorithm 11). The algorithm switches between these stages until an overall termination criterion is met. Note that we switch the cost minimisation phase off after M consecutive failures in finding a shorter tour. There are, however, two disadvantages to this approach. First, several parameters need to be tuned to allocate the budget between the two phases. More importantly, diversity maximisation is neglected during the cost minimisation phase and vice versa. This causes a negative impact on the algorithm's efficiency.

Next, we introduce a single-stage algorithm to overcome the aforementioned issues. In this algorithm, two tours p_1 and p_2 are simultaneously generated such that $c(p_1) \leq c(p_2)$ and $H(P \setminus \{p_1\}) \leq H(P \setminus \{p_2\})$. I.e. p_1 is dedicated to cost optimisation, while p_2 is generated to increase the entropy of the population. Algorithm 14 outlines the approach by means of pseudo-code. The algorithm is initialised with a population of tours optimised locally by 2-OPT. Let *Best* be the best-found solution in P. Within the evolutionary loop, two individuals p_1 and p_2 are selected uniformly at random to serve as parents and two tours are generated from these parents; one by EAX-1AB CO (p_3, focus on solution quality) and another with EAX-EDO CO (p_4, focus on diversity). p_1 is replaced with p_3 if p_3 has lower costs than *Best* or it has a lower cost than p_1 and the algorithm has not violated the M consecutive fitness evaluations without

Algorithm 12 Two-stage EAX-EDO

```
Require: Initial Population $P$, and a limit for consecutive failures in improvement
    of the shortest tour $M$.
 1: Let $Best$ be the shortest tour in $P$.
 2: $q \leftarrow 0$   // $q$ number of consecutive failures in improvement of the shortest tour
 3: while termination criterion is not met do
 4:     if $q < M$ then
 5:         $P \leftarrow$ Cost-Minimising-EA($P$).                               // Alg. 13
 6:         Let $c_{max}$ be the largest tour length in $P$.
 7:     if $c(p) < c(Best)$ then
 8:         Update $Best$
 9:         $q \leftarrow 0$
10:     else
11:         Set $q \leftarrow q + 1$
12:     $P \leftarrow$ Diversity-Maximising-EA($P, c_{max}$)                        // Alg. 11
```

Algorithm 13 Cost-Minimising-EA

```
Require: Population $P$
 1: while termination criterion is not met do
 2:     Choose randomly two individuals, $p_1, p_2 \in P$, as the parents and generate on
        offspring $p_3$ by EAX-1AB
 3:     if $c(p_3) \leq c(p_1)$ then
 4:         Replace $p_1$ with $p_3$ in $P$.
```

improvement in $Best$. Otherwise, p_4 is added to the population if $c(p_4) \leq c_{max}$. Next, if the size of the population is $\mu + 1$, the algorithm drops the individual $p \in P \setminus P^*$ whose deletion results in the smallest decrease in population diversity. A subset P^* of the best $k\%$ of the population in terms of costs always remains in the population to avoid loss of high-quality candidates until M consecutive failures in the improvement of $Best$. Eventually, P^* and c_{max} are updated and the next iteration begins. These steps continue until a termination criterion is met. Note that it is not required to generate the two offspring separately. The vast majority of necessary calculations are identical for both tours. Thus, one can simultaneously generate both tours with a single calculation to decrease computational costs.

4.4 Experimental Investigation

We perform extensive experiments in order to evaluate the introduced algorithms and the EAX-EDO CO in the settings when the optimal solution for the TSP is known and unknown.

4.4.1 Known Optimal Solution

First, we compare results where ED, PD, and H are incorporated into Algorithm 11 as the fitness function in order to select the best diversity measure. Having selected the diversity measure, we examine the performance of the operators EAX-EDO CO,

Algorithm 14 Single-stage EAX-EDO

```
Require: Initial Population $P$, and a limit for consecutive failures in the improvement
    of the shortest tour $M$.
 1: Store the $k\%$ of $P$ with shortest tours in $P^*$.
 2: Store the shortest tour in $Best$
 3: Let $c_{max}$ be the maximum cost within the population.
 4: $q \leftarrow 0$          // counts the number of consecutive failures in improvement of the
    shortest tour
 5: while termination criterion is not met do
 6:     Choose $p_1$ and $p_2 \in P$ based on parent selection procedure, produce two off-
        spring $p_3$ and $p_4$ by Crossover.
 7:     if $c(p_3) < Best$ then
 8:         Replace $p_1$ with $p_3$ in $P$ and update $P^*$ and $Best$, and set $q = 0$
 9:     else if $c(p_3) < c(p_1)$ & $q < M$ then
10:         Replace $p_1$ with $p_3$ in $P$, update $P^*$, and set $q \leftarrow q + 1$.
11:     else if $c(p_4) \leq c_{max}$ then
12:         Add $p_4$ to $P$.
13:         if q < M then
14:             Remove one individual $p$ from $P$, where $p = \arg\max_{q \in P \setminus P^*} H(P \setminus \{q\})$ and
                update $c_{max}$
15:         else
16:             Remove one individual $p$ from $P$, where $p = \arg\max_{q \in P \setminus Best} H(P \setminus \{q\})$
                and update $c_{max}$
17:         $q \leftarrow q + 1$
18:     else
19:         Set $q \leftarrow q + 1$
```

EAX-1AB and 2-OPT. For this purpose, we conduct experiments for all combinations of $\mu \in \{50, 100\}$ and $\alpha \in \{0.05, 0.1, 0.5\}$ on instances eil51, eil76, and eil101 from the TSPlib [97] for 10 independent runs.

Comparison between diversity measures

In this subsection, we examine the performance of Algorithm 11 in the case where H, ED, and PD are embedded into the algorithm as the fitness function. The algorithm is initialised with μ copies of an optimal solution, and 2-OPT is used as the operator to generate offspring.

TABLE 4.1. Comparison of diversity measures. In columns stat, the notation X^+ means the median of the measure is better than the one for variant X, X^- means it is worse and X^* indicates no significant difference. Stat indicates the results of the Kruskal-Wallis statistical test at the significance level 5% and Bonferroni correction.

			ENT (1)						**ED** (2)						**PD** (3)					
	μ	α	**H**	stat	**ED**	stat	**PD**	stat	**H**	stat	**ED**	stat	**PD**	stat	**H**	stat	**ED**	stat	**PD**	stat
eil51	50	0.05	**0.60**	2^+3^+	**0.35**	2^*3^+	0.05	2^*3^-	0.51	1^-3^-	0.32	1^*3^+	0.02	1^-3^-	0.52	1^-2^+	0.28	1^-2^-	**0.18**	1^+2^+
eil51	50	0.1	**0.86**	2^+3^+	**0.47**	2^*3^+	0.12	2^+3^-	0.73	1^-3^-	0.46	1^*3^+	0.04	1^-3^-	0.79	1^-2^+	0.41	1^-2^-	**0.30**	1^+2^+
eil51	50	0.5	**1.79**	2^+3^+	0.77	2^-3^*	0.55	2^+3^-	1.60	1^-3^-	**0.81**	1^+3^+	0.20	1^-3^-	1.75	1^-2^+	0.78	1^*2^-	**0.71**	1^+2^+
eil51	100	0.05	**0.60**	2^+3^+	**0.34**	2^+3^+	0.05	2^*3^-	0.50	1^-3^-	0.29	1^-3^+	0.02	1^-3^-	0.51	1^-2^+	0.26	1^-2^-	**0.15**	1^+2^+
eil51	100	0.1	**0.88**	2^+3^+	**0.47**	2^+3^+	0.08	2^+3^-	0.73	1^-3^-	0.44	1^-3^+	0.03	1^-3^-	0.76	1^-2^+	0.37	1^-2^-	**0.24**	1^+2^+
eil51	100	0.5	**1.81**	2^+3^+	0.77	2^-3^+	0.39	2^+3^-	1.59	1^-3^-	**0.80**	1^+3^+	0.10	1^-3^-	1.77	1^-2^+	0.77	1^-2^-	**0.65**	1^+2^+
eil76	50	0.05	**0.51**	2^+3^+	**0.28**	2^+3^+	0.06	2^*3^-	0.43	1^-3^-	0.24	1^-3^+	0.03	1^*3^-	0.43	1^-2^+	0.22	1^-2^-	**0.15**	1^+2^+
eil76	50	0.1	**0.77**	2^+3^+	**0.41**	2^+3^+	0.11	2^+3^-	0.68	1^-3^-	0.38	1^-3^+	0.05	1^-3^-	0.68	1^-2^+	0.33	1^-2^-	**0.25**	1^+2^+
eil76	50	0.5	**1.78**	2^+3^+	0.75	2^-3^+	0.55	2^+3^-	1.62	1^-3^-	**0.8**	1^+3^+	0.13	1^-3^-	1.74	1^-2^+	0.75	1^*2^-	**0.68**	1^+2^+
eil76	100	0.05	**0.50**	2^+3^+	**0.26**	2^+3^+	0.05	2^*3^-	0.41	1^-3^-	0.22	1^-3^+	0.03	1^-3^-	0.41	1^-2^+	0.19	1^-2^-	**0.11**	1^+2^+
eil76	100	0.1	**0.76**	2^+3^+	**0.38**	2^+3^+	0.08	2^*3^-	0.65	1^-3^-	0.34	1^-3^+	0.04	1^-3^-	0.64	1^-2^+	0.29	1^-2^-	**0.19**	1^+2^+
eil76	100	0.5	**1.79**	2^+3^+	0.74	2^-3^+	0.35	2^+3^-	1.63	1^-3^-	**0.79**	1^+3^+	0.07	1^-3^-	1.72	1^-2^+	0.69	1^-2^-	**0.56**	1^+2^+
eil101	50	0.05	**0.52**	2^+3^+	**0.28**	2^+3^+	0.07	2^*3^-	0.45	1^-3^-	0.24	1^-3^+	0.05	1^*3^-	0.43	1^-2^+	0.21	1^-2^-	**0.15**	1^+2^+
eil101	50	0.1	**0.75**	2^+3^+	**0.39**	2^+3^+	0.11	2^+3^-	0.67	1^-3^-	0.37	1^*3^+	0.05	1^-3^-	0.66	1^-2^+	0.31	1^-2^-	**0.23**	1^+2^+
eil101	50	0.5	**1.76**	2^+3^+	0.72	2^-3^+	0.53	2^+3^-	1.56	1^-3^-	**0.76**	1^+3^+	0.08	1^-3^-	1.71	1^-2^+	0.71	1^-2^-	**0.61**	1^+2^+
eil101	100	0.05	**0.49**	2^+3^+	**0.24**	2^+3^+	0.05	2^*3^-	0.42	1^-3^-	0.21	1^*3^+	0.03	1^*3^-	0.4	1^-2^+	0.18	1^-2^-	**0.11**	1^+2^+
eil101	100	0.1	**0.72**	2^+3^+	**0.35**	2^+3^+	0.06	2^*3^-	0.62	1^-3^-	0.32	1^-3^+	0.04	1^-3^-	0.58	1^-2^+	0.25	1^-2^-	**0.17**	1^+2^+
eil101	100	0.5	**1.74**	2^+3^+	0.70	2^-3^+	0.23	2^+3^-	1.54	1^-3^-	**0.74**	1^+3^+	0.04	1^-3^-	1.61	1^-2^+	0.62	1^-2^-	**0.53**	1^+2^+

Table 4.1 shows that the algorithm using entropy H as the fitness function not only outperforms its counterparts in terms of entropy value (H) in all the cases. It also leads to higher ED scores compared to the algorithm using ED as the fitness function when α is equal to 0.05 or 0.1. Moreover, the entropy-based algorithm (H) results in higher PD scores compared to the ED-based algorithm and higher ED scores in comparison to the PD-based algorithm. These observations are supported by the results of the Kruskal-Wallis test at significance level 5% and Bonferroni correction. One underlying reason for observing these significant differences in the results is that the algorithms have different fitness functions, which have been used alternatively in these statistical tests. The entropy-based measure is selected as the fitness function for the following experiments.

Comparison between EAX-EDO CO, EAX-1AB CO, 2-OPT

Here, we investigate the performance of the proposed EAX-EDO CO in comparison to EAX-1AB and 2-OPT. Note that we used 2-OPT in the first 1 000 iterations for all competitors since EAX-EDO CO and EAX-1AB as crossover operators require two different parents to generate an offspring that is no clone, while the initial population consists of μ copies of a single solution. The experimental settings and instances are in line with the previous subsection.

Figure 4.2 shows the performance of EAX-EDO CO, EAX-1AB CO, and 2-OPT (encapsulated in Algorithm 11). The figure illustrates that not only does EAX-EDO CO outperform EAX-1AB CO and 2-OPT in terms of the mean diversity score in all cases, but also has a lower standard deviation. These observations are confirmed by the results of a Kruskal-Wallis test at significance level 5% and Bonferroni correction that indicates a significant difference is found in the median of entropy scores of the final populations obtained by EAX-EDO CO and that of the EAX-1AB and 2-OPT in all considered settings. Turning to the comparison between EAX-1AB and 2-OPT, the former brings about more diverse populations in most of the cases, except for the setting with $\alpha = 0.5$ and $\mu = 50$ on eil51. One can notice that the smaller α is, the larger is the gap between EAX-based crossovers and 2-OPT. This is because 2-OPT is a random neighbourhood search; therefore, the tighter the threshold is, the lower the chance of generating an offspring individual with acceptable quality. Note that a larger neighbourhood search would be necessary to escape local optima.

4.4.2 Unknown Optimal Solution

The two-stage EAX-EDO introduced for unknown optimal cases includes several parameters, such as parameters associated with the allocation of the budget between the cost minimisation and diversity maximisation phases. It is crucial to tune the parameters in order to deliver the algorithm's best performance.

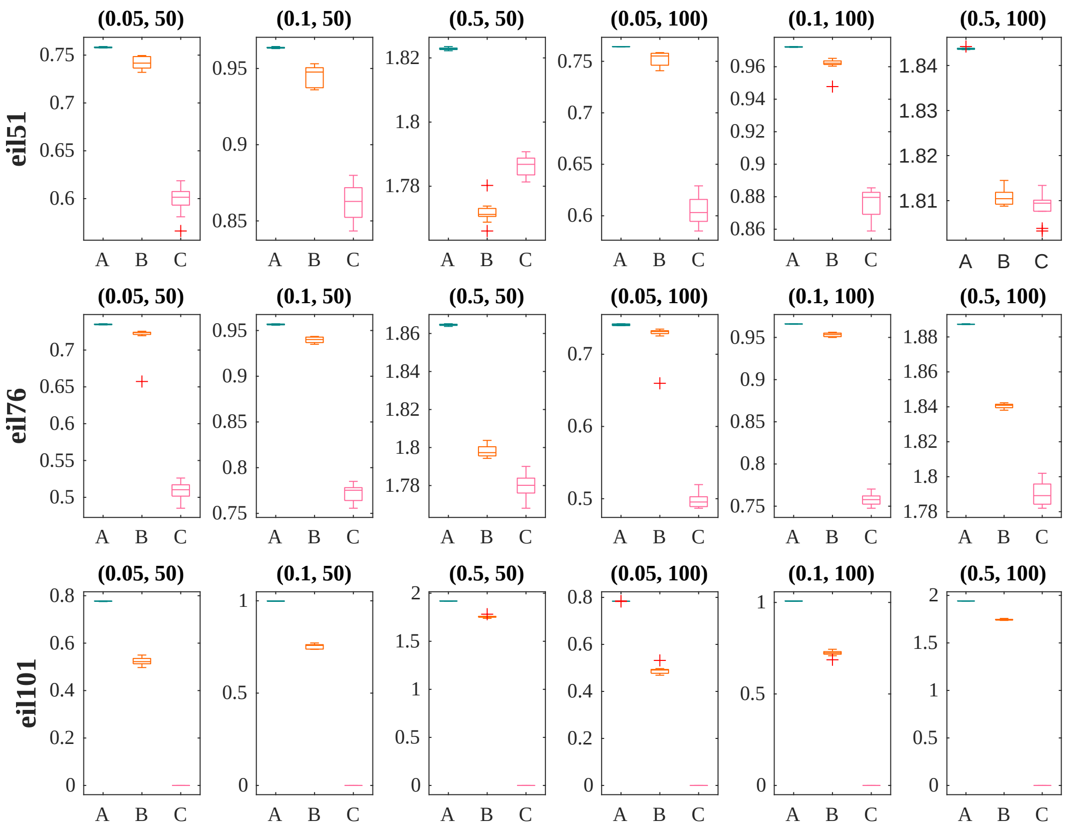


FIGURE 4.2. Distributions of diversity scores (to be maximised) of populations obtained by Algorithm 11 with EAX-EDO CO (A), EAX-1AB CO (B), and 2-OPT (C) for instances eil51, eil76, and eil101. Labels above the box plots indicate (α, μ).

Algorithm Configuration

The goal of automatic algorithm configuration is to find, in an automated fashion, a parameter configuration to deliver the algorithms' best (average) performance for a given set of instances. Here, the algorithm parameters are tuned by iRace [61], which performs an iterated racing procedure between different parameter configurations to find the best parameter setting. In our experimental investigation, we consider the minimum budget of 96 runs to be performed by iRace for each algorithm due to the fact that the algorithms are computationally expensive.

The two-stage algorithm includes four input parameters. Since the budget is limited, there are three parameters associated with the allocation of the budget, which include budget allocation to each repetition of the main loop (X), and the proportions of X are allocated to cost minimisation and diversity maximisation phases, x_c and x_d, respectively. The last parameter is the number of consecutive failures M of the cost minimisation phase in finding a tour with better costs. Note that there exist dependencies between these parameters. First, X is a proportion of the total budget (It). Second, x_d can be determined as $1 - x_c$. Finally, M should be lower than the total number of repetitions of the main loop, i.e. $M_1 < It/X$; therefore, we set

Table 4.2. Tested parameter values during the tuning procedure.

Parameter	X	x_c	m
Range	$(0.03, 0.2)$	$(0.2, 0.8)$	$(0.1, 0.5)$
Best setting	0.0614	0.4888	0.2413

$M = m \cdot (It/X)$, where $m \in (0, 1)$.

Table 4.2 shows the parameter ranges considered in the tuning procedure and the best setting found by iRace. The other parameters can be calculated from the aforementioned equations.

Experiments

We now compare the performance of single-stage and two-stage EAX-EDO against standard EAX and the Gurobi optimiser [45] in terms of solution quality and entropy-based diversity. The Gurobi optimizer is a well-known mixed integer programming (MIP) solver. Although Gurobi is usually used to obtain the optimal or a high-quality solution for a given optimisation problem, it is capable of providing its users with μ different solutions within a specific gap α to the optimal solution. Here, we use the Dantzig–Fulkerson–Johnson formulation [23]. We use Gurobi to generate μ different solutions and use it as a baseline for comparison in our studies.

The benchmark instances considered in this section include eil101, a280, pr493, u574, rat575, p654, rat783, u1060, pr2392, and fnl4461 [97]. Moreover, we set μ to 50. Note that all algorithms are initialized with a population of individuals optimised by 2-OPT. The results are summarised in Table 4.3.

The outcome indicates that the introduced single-stage EAX-EDO outperforms the other algorithms in terms of diversity. Table 4.3 shows that both single-stage EAX-EDO and two-stage EAX-EDO are capable of computing tours with decent costs. The Gurobi optimiser and EAX result in marginally better quality, while they are outperformed by the single-stage EAX-EDO in terms of diversity in all instances except on instance fnl4461. In this instance, single-stage EAX-EDO leads to a higher quality but less diverse population compared to EAX.

This can be attributed to EAX generating $\lambda = 25$ offspring per iteration, which makes it converge slower than the two variants of EAX-EDO. Therefore, the EAX-EDO algorithms achieve better quality but less diverse populations in 500 000 fitness evaluations. Should EAX continue to run, we expect that it converges to a slightly higher quality but less diverse population compared to EAX-EDO; the same trend can be observed for smaller instances. Figure 4.3 illustrates this matter visually. The figure depicts the quality of best solutions and diversity of populations over fitness evaluations for the single-stage EAX-EDO, the two-stage EAX-EDO, and vanilla

EAX. Figure 4.3 shows that having a lower entropy in preliminary iterations, single-stage EAX-EDO finally converges to higher entropy compared to the vanilla EAX on rat783, u1060, pr2392. It is the other way around for the quality of tours; EDO-based algorithms converge faster in terms of quality, while vanilla EAX results in slightly higher-quality tours. The same patterns can be observed for other instances except fnl4461. In this instance, none of the algorithms converges in this setting. Here, Figure 4.3 indicates that 500 000 fitness evaluations are insufficient to have the algorithms converged.

TABLE 4.3. Comparison of the proposed algorithms with EAX in terms of diversity (H) and solution quality of the best solution (c). Here, ΔH shows the entropy of the final population P on top of H_{min}, i.e, $\Delta H = H(P) - H_{min}$. Columns $stat_H$ and $stat_c$ contain the results of Kruskal-Wallis tests on the entropy of the final population and best tour length, respectively.

			Gurobi		**1-stage EAX-EDO** (1)				**2-stage EAX-EDO** (2)				**EAX** (3)			
	H_{min}	OPT	ΔH	c	ΔH	c	stat$_H$	stat$_c$	ΔH	c	stat$_H$	stat$_c$	ΔH	c	stat$_H$	stat$_c$
eil101	5.31	629	0.11	629	**0.79**	629	2^+3^+	2^*3^*	0.34	629	1^-3^+	1^*3^*	0.11	629	1^-2^-	1^*2^*
a280	6.33	2 579	0.1	2 579	**0.6**	2 579	2^+3^+	2^*3^*	0.30	2 579	1^-3^+	1^*3^*	0.12	2 579	1^-2^-	1^*2^*
pr439	6.78	107 217	0.03	107 217	**0.66**	107 262	2^-3^*	2^-3^*	0.27	107 217	1^*3^+	1^+3^*	0.05	107 226	1^-2^-	1^*2^*
u574	7.05	36 9054	0.02	36 905	**0.67**	36 914	2^+3^+	2^-3^-	0.29	36 908	1^-3^+	1^*3^*	0.01	36 905	1^-2^-	1^*2^*
rat575	7.05	6 773	0.05	6 773	**0.63**	6 777	2^+3^+	2^-3^-	0.35	6 775	1^-3^+	1^*3^*	0.11	6 774	1^-2^-	1^+2^*
p654	7.18	34 643	0.08	34 643	**1.15**	34 643	2^+3^+	2^*3^*	0.55	34 646	1^-3^+	1^*3^-	0.28	34 643	1^-2^-	1^*2^+
rat783	7.36	8 806	0.02	8 806	**0.57**	8 809	2^+3^+	2^*3^-	0.31	8 807	1^-3^+	1^*3^*	0.07	8 806	1^-2^-	1^+2^*
u1060	7.66	224 094	-	-	**0.69**	224 275	2^+3^+	2^*3^-	0.35	224 131	1^-3^+	1^*3^*	0.07	224 109	1^-2^-	1^+2^*
pr2392	8.47	378 032	-	-	**0.56**	378 813	2^+3^+	2^*3^-	0.28	378 926	1^-3^+	1^*3^-	0.02	378 059	1^-2^-	1^+2^+
fnl4461	9.1	182 566	-	-	0.33	182 297	2^*3^-	2^*3^+	0.32	183 200	1^+3^-	1^-3^+	**0.42**	184 230	1^+2^+	1^-2^-

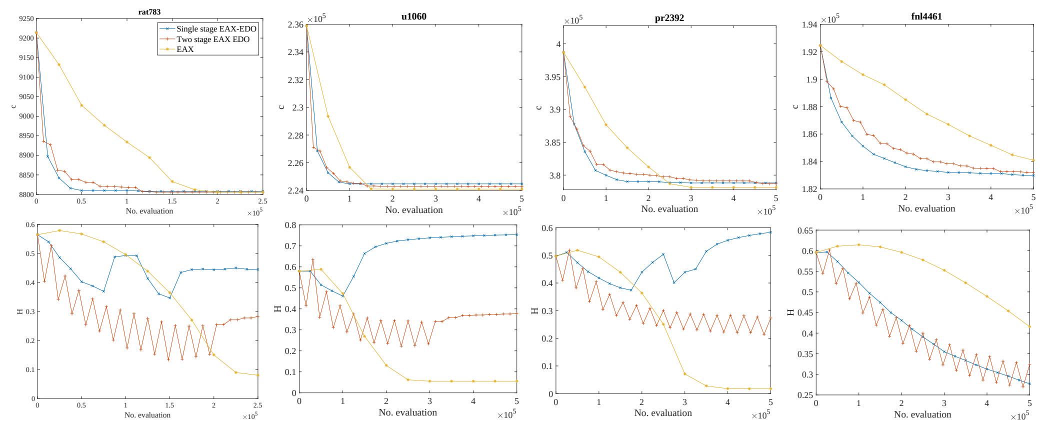


Figure 4.3. Representative trajectories in the setting of unknown optimal solutions. The plots show the best tour length in the population (first row) and diversity measured by the entropy (second row).

Moreover, Figure 4.3 shows that EAX sacrifices the entropy of the population to gain shorter tours. On the contrary, single-stage EAX-EDO increases the entropy while in the course of optimising the solution quality. In addition, Figure 4.3 highlights the room for improvement in the EAX-EDO algorithms. First, tuning the single-stage EAX-EDO's parameters is likely to boost the performance of the algorithm, although it outperforms the other counterparts in terms of diversity in the current state. Second, the EAX-EDO algorithms generate one offspring per iteration; generating λ number of offspring from the same parents and incorporating the selection procedure to choose between them (same as the standard EAX), EAX-EDO is likely to achieve even higher diversity and quality. However, the selection of μ tours from $\mu+\lambda$ tours in a way that maximises diversity is a complicated problem. For instance, there are $\binom{\mu+\lambda}{\mu}$ possible candidates for Brute-force search. This makes the algorithm computationally more expensive. We already investigated $\lambda \geq 2$ in the EDO algorithm designed for scenarios where the optimal tour is known a priori in Section 3.7.

Figure 4.4 visualises exemplary populations obtained by the single-stage and the two-stage EAX-EDO, Gurobi and EAX. The figure aids in comprehending how populations obtained by the EAX-EDO algorithms differ from the ones computed by standard EAX and Gurobi. As one can notice from Figure 4.4, the single-stage EAX-EDO incorporates a higher number of edges into the population compared to the other algorithms. For example, on eil101, the population obtained from the single-stage EAX-EDO includes 758 unique edges, while the number of edges for two-stage EAX-EDO, Gurobi and EAX are 416, 238, and 238, respectively. A similar pattern can be observed in all the other instances; this includes a280 and u575, as it is shown in Figure 4.4. Moreover, the figure depicts that EAX and Gurobi are almost incapable of having low frequent edges in the populations. However, the two EDO frameworks, especially those of the single-stage EAX-EDO, incorporate many low frequent edges into the populations.

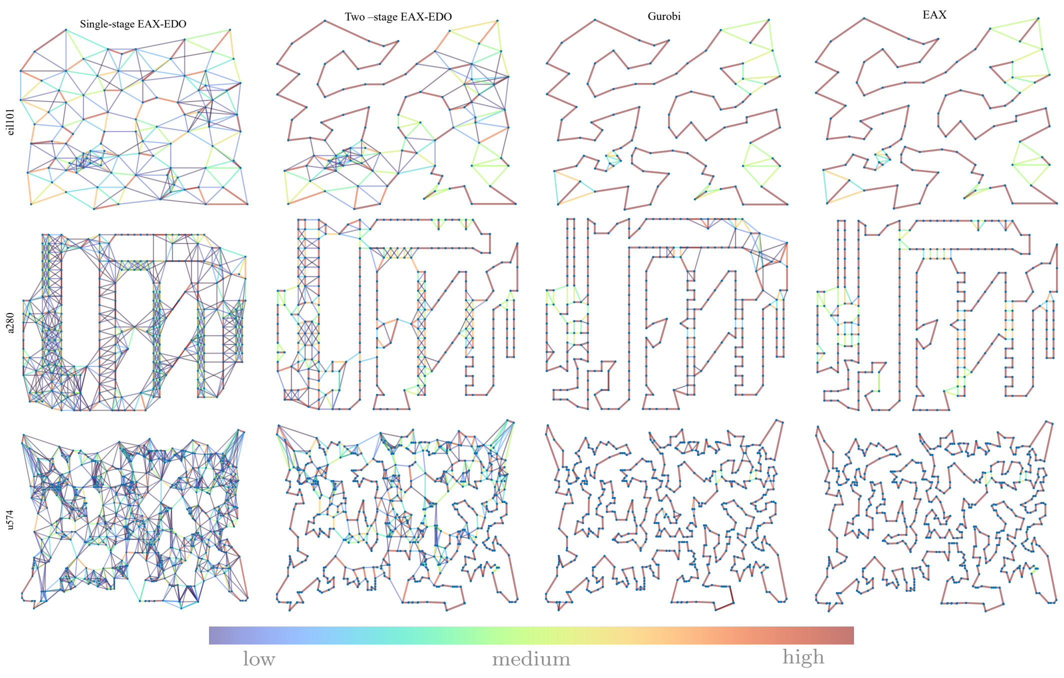


FIGURE 4.4. Overlay of all edges used in exemplary final populations. Edges are coloured by their frequency.

Robustness of the populations

One motivation for EDO is, as stated earlier that decision-makers can choose between different alternatives if they are provided with a diverse set of high-quality solutions. For instance, decision-makers can avoid a certain edge if they prefer to, or the edge becomes unavailable for some reason. In this section, we compare the robustness of the population obtained from the four competitors when one or more edges of the optimal tour for a given TSP instance suddenly become unavailable. To this end, we randomly make one, two, and three edges of the optimal solution unavailable. Next, we determine 1) the percentage of occasions (over 1000 independent experiments) where there is at least one alternative tour in the population (encoded by a) and 2) the mean of different alternative tours in populations that avoid those edges (encoded by d). Table 4.4 summarises the results of this series of experiments.

The outcome indicates that the two variants of EAX-EDO lead to more robust populations against minor changes compared to EAX and Gurobi. In fact, the single-stage EAX-EDO has superior performance compared to its competitors: its population, e.g. on a280, succeeds in 83% of occasions to offer an alternative when an edge becomes unavailable, whereas the Two-stage EAX-EDO scores 55%, the EAX scores 23% and the Gurobi achieves 26%, respectively. Moreover, the population of the single-stage EDO includes 5.8 alternative tours to the optimal on average when two edges become unavailable. This is while this figure is 1.69 for the Two-stage EAX-EDO, and the populations of EAX and the Gurobi optimiser are incapable of offering any alternative on two and three instances, respectively. In case three edges are eliminated, the standard EAX and Gurobi are barely able to offer any alternatives, while success rates of the single-stage and the two-stage EAX-EDO are 42% and 8.86%, respectively. Therefore, considering Table 4.4, we can claim that the single-stage EAX-EDO framework outperforms the classic and the two-stage EDO frameworks in terms of population robustness.

Table 4.4. Comparison of the robustness of the populations obtained from single-stage EAX-EDO (1), two-stage EAX-EDO (2), EAX (3), and Gurobi (4) in case one, two, or three random edges from the optimal solution become unavailable in 100 runs. a denotes the percentage of times the population has at least one alternative for the eliminated edges, while d represents the number of alternative tours avoiding the eliminated edges on average.

	One edge								**Two edges**								**Three edges**							
	EDO (1)		**EDO (2)**		**EAX (3)**		**Gurobi (4)**		**EDO (1)**		**EDO (2)**		**EAX (3)**		**Gurobi (4)**		**EDO (1)**		**EDO (2)**		**EAX (3)**		**Gurobi (4)**	
	a	d	a	d	a	d	a	d	a	d	a	d	a	d	a	d	a	d	a	d	a	d	a	d
eil101	**90**	**18.07**	50	9.5	18	4.34	18	3.79	**74**	**6.57**	20	1.82	3	0.33	3	0.27	**50**	**2.29**	8	0.43	1	0.04	0	0.03
a280	**83**	**15.15**	55	7.58	23	3.83	26	1.89	**64**	**5.03**	21	1.21	5	0.34	3	0.18	**40**	**1.58**	7	0.25	1	0.04	0	0.01
pr439	**82**	**14.95**	36	5.05	0	0	10	0.70	**58**	**4.41**	7	0.40	0	0	0	0.01	**30**	**1.20**	2	0.07	0	0	0	0
u574	**88**	**15.37**	46	6.79	1	0.27	6	0.42	**65**	**5.08**	15	1.13	0	0	0	0.01	**39**	**1.58**	5	0.252	0	0	0	0
rat575	**85**	**14.40**	61	9.06	26	4.64	21	0.94	**57**	**4.08**	29	1.92	4	0.391	1	0.02	**33**	**1.30**	9	0.36	1	0.04	0	0
p654	**90**	**23.84**	61	14.19	55	10.24	33	8.20	**79**	**11.48**	39	4.15	27	2.16	9	1.89	**70**	**6.08**	24	1.28	11	0.49	2	0.26
rat783	**82**	**13.54**	55	7.33	19	2.02	4	0.65	**56**	**3.98**	19	1.18	3	0.12	0	0.016	**32**	**1.24**	7	0.21	1	0.006	0	0

4.5 Conclusion

In this chapter, we introduced EAX-based evolutionary diversity optimisation approaches for the well-known TSP, which are able to compute diverse sets of high-quality TSP tours. We designed an entropy-based diversity measure for the TSP and modified the powerful EAX operator towards a variant called EAX-EDO crossover that allows to simultaneously minimise the tour length and maximise the population diversity. The resulting EAX-EDO algorithms allow computing high-quality, diverse sets of TSP tours through a two-stage approach that alternates between optimised tour lengths and the diversity of the population or a single-stage method that optimises both criteria simultaneously. Our experimental results show that (1) EAX-EDO crossover outperforms recent approaches from the literature based on k-OPT neighbourhood search in a setting where the optimal tour is known and (2) the introduced algorithms show superior performance with respect to diversity while being competitive with respect to the objective function in comparison with the pure EAX and the Gurobi optimiser on a subset of classical TSP benchmark instances when an optimal solution is unknown. Moreover, our results indicate that EAX-EDO algorithms compute more robust populations compared to the classic optimisation frameworks.

Future work will focus on the enhancement of EAX-EDO in terms of (1) generating several offspring in each iteration and (2) a method to select a population from a pool of new offspring and old individuals. We expect these modifications to boost the performance of EAX-EDO in terms of both quality and diversity. Moreover, it is intriguing to investigate the application of EDO on real-world problems.

Chapter 5

Quality Diversity Algorithms for the Traveling Thief Problem

5.1 Introduction

We studied TSP in the context of EDO in previous chapters. We now investigate diversity in TTP solutions in the following two chapters, starting with QD algorithms. QD has been shown to be very powerful in diversifying solutions in terms of behavioural properties. It is also capable of efficiently illustrating the distribution of high-quality solutions in behavioural space. The use of QD in continuous optimisation problems has been investigated extensively and shown that the QD-based algorithm yields decent results.

The decent results of QD algorithms in the continuous domain and the lack of study in the context of combinatorial optimisation problems motivate us to introduce a QD-based algorithm in a problem in the discrete domain, the TTP. QD is a powerful tool to maintain diversity among solutions, and we believe it can result in TTP solutions with decent results.

We employ the concept of QD for solving the TTP. By this means, we scrutinise the distribution of high-performing TTP solutions in the behavioural space of the TSP and the KP and compute very high-quality solutions. We introduce a bi-level MAP-Elite-based evolutionary algorithm called BMBEA. The algorithm generates new solutions in a two-stage procedure. First, it generates new high-quality TSP tours from old ones by the well-established EAX crossover operator [72] for the TSP or as an alternative by 2-OPT [20]. Second, it utilises dynamic programming [81] or an (1+1) evolutionary algorithm to compute an optimal (or near-optimal) packing list for the given TSP tour. Having generated a new solution, BMBEA applies a MAP-Elites-based survival selection to achieve a diverse set of high-quality TTP solutions. To achieve diversity, MAP-Elites is applied with respect to the two-dimensional space given by the TSP and KP quality of the TTP solutions. To form such as behavioural space, we should know the optimal values for the sub-problems. Here, we use EAX [72] and DP [111] to compute those values.

We conduct a comprehensive experimental investigation to analyse and visualise the distribution of high-quality TTP solutions for different TTP instances. Furthermore, we show the capability of BMBEA to generate high-performing TTP solutions. The algorithm results in very high TTP values and improves the best-known TTP solution for some benchmark instances. Moreover, we propose a method that eliminates two influential input parameters that need to be tuned for each instance individually. Using the method requires a higher number of generations to converge compared to the previous method with the considered input values. However, it improves the results, especially for the larger instances. We also investigate the impact of Map-Elite-based survival selection. We show that the survival selection brings about a diverse set of solutions that prevents premature convergence.

The work of this chapter is based on a conference paper [88] presented at the genetic and evolutionary computation conference (GECCO 2022) and its extended version that is submitted to ACM Transactions on Evolutionary Learning and Optimization journal. We also correct some incorrect experimental results from the conference version, which were due to an implementation error.

The remainder of this chapter is structured as follows. In Section 5.2, we formally define the TTP problem. We introduce the MAP-Elites-based approach for TTP and the BMBEA algorithm in Section 5.3. We also propose a baseline algorithm to investigate the impact of MAP-Elitism in Section 5.3. We examine the high-quality TTP solutions in terms of their TSP and KP score and report on our results using BMBEA for solving the TTP are shown in Section 5.4. Finally, we finish with some concluding remarks.

5.2 The Traveling Thief Problem

We formally defined the TTP in Chapter 2.4.3. To ease the reading, we shortly redefine it here as well. As mentioned, TTP is formed by the integration of the TSP and the KP. The TTP is formed by a graph G and a set of items I where items are equally scattered on the cities. Each city i except the first one contains a set of items M_i (a subset of I), and each item k located in the city i is associated with a profit p_{ik} and a weight w_{ik}. A TTP solution (x, y) consists of a tour x and a packing list y that maximises:

$$z(x,y) = \sum_{j=1}^{m} p_j y_j - R\left(\frac{d_{x_n x_1}}{\nu_{max} - \nu W_{x_n}} + \sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{\nu_{max} - \nu W_{x_i}}\right)$$
$$\text{subject to } \sum_{j=1}^{m} w_j y_j \leq W.$$

Here, $\nu_{\max}$ and $\nu_{\min}$ are the maximal and minimal travelling speed, $\nu = \frac{\nu_{max}-\nu_{min}}{W}$ is a constant, and W_{x_i} is the cumulative weight of the items collected from the start of the tour up to city x_i. In this chapter, $z(x, y)$ serves as the fitness function, while $f(x)$, the TSP function and $g(y)$ the KP function serve as the behavioural descriptor (BD):

$$f(x) = d(x(n), x(1)) + \sum_{i=1}^{n-1} d(x(i), x(i+1)).$$

$$g(y) = \sum_{j=1}^{m} p_j y_j$$
$$\text{subject to } \sum_{j=1}^{m} w_j y_j \leq W.$$

Generally, the fitness function indicates how well a solution solves the given problem, while the BD shows how it solves the problem and behaves in terms of the features. In this case, the BD presents the length of the tour (f) and the value of items collected (g), whereby the fitness function returns the overall profit (z). Here, we aim to compute a diverse set of high-quality solutions differing in the BD. By this means, we can look into the distribution of high-performing TTP solutions over the 2D space of TSP and KP.

5.3 Bi-level Map-Elites-based Evolutionary Algorithm

Map-Elites is an evolutionary computation approach where solutions compete with each other to survive. However, competition is only among solutions with a similar BD value in order to maintain diversity. We require a hyperparameter to define the similarity and the tolerance of acceptable differences between two descriptors. In the MAP-Elites algorithms, the BD space is discretised into a grid, where each cell is associated with one BD type. It means each solution belongs to at most one cell in the behavioural space (the map). Map-Elite algorithms typically keep only the best solution in each cell. When a solution is generated, it is assessed and potentially added to the cell with the associated BD. If the cell is empty, the solution occupies the cell; otherwise, the best solution is kept in the cell. The map aids in understanding and

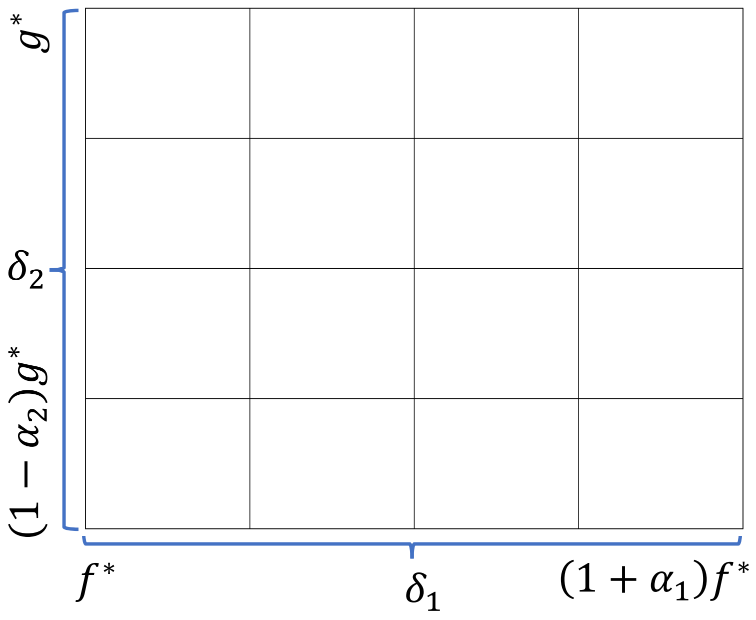


FIGURE 5.1. The representation of an empty map. There are $\delta_1 \times \delta_2$ cells within the map.

visualising the distribution of high-quality TTP solutions. For instance, how much should we move away from the optimal TSP tour and the optimal KP solution to generate high-performing TTP solutions?

Generally, the behavioural space can be extremely large. Thus, it is rational to limit the map to a promising part of the space; otherwise, either the number or the size of the cells increases severely, and as a result, the performance and efficiency of the algorithms is undermined. As mentioned, a TTP solution consists of a tour and a packing list that belong to the TSP and the KP components of the problem. Although solving each sub-problem separately does not necessarily result in a high-quality TTP solution, a TTP solution should score fairly well in both features in order to gain high profits. Thus, we focus on solutions within α_1 and α_2 gap to the optimal TSP value (f^*) and the optimal KP value (g^*), respectively. In this chapter, α_1 and α_2 are set to 5 and 20, respectively, based on initial experimental investigations. Figure 5.1 depicts an empty map. There are $\delta_1 \times \delta_2$ cells. Cell (i, j), $1 \leq i \leq \delta_1$, $1 \leq j \leq \delta_2$ contains the best found solution with TSP score in $\left[f^* + (i-1)\left(\frac{\alpha_1 f^*}{\delta_1}\right), f^* + (i)\left(\frac{\alpha_1 f^*}{\delta_1}\right)\right)$ and KP score in $\left[(1-\alpha_2)g^* + (j-1)\left(\frac{\alpha_2 g^*}{\delta_2}\right), (1-\alpha_2)g^* + (j)\left(\frac{\alpha_2 g^*}{\delta_2}\right)\right)$, the corresponding BD. The cell (1, δ_2) consists of TTP solutions with TSP and KP values closest to the optimums. In this chapter, we require to know f^* and g^*. For this purpose, we can use EAX [72] and dynamic programming (DP) [111] to compute f^* and g^* for the TTP instances, respectively.

Algorithm 15 describes the BMBEA. The initialising procedure and the operators to generate a new TTP solution will be discussed later. Having generated an empty map, we populate it with an initialising procedure. After generating offspring, we

Algorithm 15 The MAP-Elites-Based Evolutionary Algorithm

```
1: Find the optimal/near-optimal values of the TSP and the KP by algorithms in
   [72, 111], respectively.
2: Generate an empty map and populate it with the initialising procedure.
3: while termination criterion is not met do
4:    Generate an offspring and calculate the TSP and the KP scores.
5:    if The TSP and the KP scores are within $\alpha_1$%, and $\alpha_2$% gaps to the optimal
      values of BD. then
6:       Find the corresponding cell to the TSP and the KP scores.
7:       if The cell is empty then
8:          Store the offspring in the cell.
9:       else
10:         Compare the offspring and the individual occupying the cell and store the
            best individual in terms of TTP score in the cell.
```

calculate the TSP score and the KP score of the offspring. If the TSP and the KP scores are within α_1% and α_2% gap of the optimal values, respectively, we find the cell corresponding to those scores; otherwise, the offspring is discarded. If the corresponding cell is empty, the offspring is kept in the cell; otherwise, we compare the offspring and the individual in the cell and keep the individual with the highest TTP score. We repeat steps 3 to 10 until a termination criterion is met.

Evolutionary algorithms require some operators to generate new solutions (offspring) from old ones (parents); BMBEA is no exception. One can see the generating of TTP solutions as a bi-level process. First, new tours can be generated by mutation or crossovers; then, we can compute a suitable packing list for the new tours to have complete TTP solutions.

5.3.1 Search Operators for TSP

We consider EAX crossover [72] to generate new TSP tours. As mentioned in Section 2.4.2, EAX is a highly performing TSP crossover known as one of the state-of-the-art operators in solving TSP. The use of EAX has also been shown to lead to high-quality solutions for the TTP in [120]. EAX has several variants; we incorporate the EAX-1AB due to its simplicity and efficiency. We already described the EAX in section 2.4.2. Alternatively, we can employ 2-OPT to generate TSP tours.

5.3.2 Search Operators for KP

In the second phase, we optimise the packing list to match the TSP tour and form a good TTP solution. To this means, inner algorithms are required to optimise the packing list. When the tour is fixed and the packing list is optimised, the problem is referred to as the packing while travelling problem (PWT). [93] in the literature. Neumann et al. [81] introduced a DP algorithm (introduced in Section 2.4) to solve the PWT problem to optimality. Here, we use the DP to obtain the optimal packing selection for the tour generated by the TSP operators.

$(1+1)$ Evolutionary Algorithm

The $(1+1)$ *EA* is a well-known simple EA that we may hope to converge fast since it only keeps the best-found solution. First, the new tour generated by the TSP operators inherits its parent's packing list. Next, a new packing list is generated by mutation. If the new packing list results in a higher TTP score, the new packing list is replaced with the old one. We continue these steps until a termination criterion is met. For mutation, the bit-flip is used, where each bit is independently flipped by mutation rate $\frac{1}{m}$.

The mutation can result in packing lists violating the knapsack's capacity. We incorporate a repair function into the $(1+1)$ *EA* to avoid the violation. After the offspring is mutated, the repair function fixes the offspring's violation. The repair function removes collected items uniformly at random one by one until the packing list complies with the capacity constraint.

5.3.3 Initialisation

One may notice that it is doubtful to populate the map with random solutions. This is because the map only accepts TTP individuals with fairly good TSP and KP scores. Therefore, a heuristic approach is required to populate the map initially. We can use the EAX-based algorithm in [72] to find the optimal/near-optimal TSP tours in terms of length. Having extracted the tours, we can compute a good quality packing list for each tour by one of the KP operators mentioned in section 5.3.2. This results in TTP solutions with high TSP and KP scores. Let us denote the set of solutions by P_0. This initial population enables us to populate the map at the beginning of the BMBEA. In this study, we use a target length (f^*) as a termination criterion for the EAX-based algorithm in [72] so as to increase the time efficiency and diversity of tours. If we do not have the target values, it is important to tune the running time of the algorithm for each instance individually. Note that in the case of using $(1+1)$EA as the KP operator, it will boost the performance of BMBEA if we start with the optimal packing plan obtained by [111].

5.3.4 A More Relaxed Map

Treating α_1 and α_2 as input required intensive tuning and preliminary experimental investigations. Randomly selecting values for α_1 and α_2 not only affects the algorithm's efficiency, but it may also bring about an infeasible map that is impossible to fill. As a result, the algorithm cannot produce any solution in such cases. Therefore, α_1 and α_2 need to be set carefully, and sometimes they should be re-tuned for different instances. To address this limitation, we propose a relaxed method in which we set the thresholds after the initialisation. Having computed P_0 as mentioned above, we set the $\alpha_1 = \left(\frac{f_{max}}{f^*}\right) - 1$ and $\alpha_2 = 1 - \left(\frac{g_{min}}{g^*}\right)$, where f_{max} and g_{min} are the largest tour and the minimum packing value in P_0, respectively.

5.3.5 $(\mu + 1)$ EA

We require a similar algorithm with a conventional survival selection to investigate the impact of QD and MAP-Elitism on the results. We consider $(\mu + 1)$EA since it has the same offspring size as the introduced algorithm. Here, we generate an initial population P_0, same as Section 5.3.3. The parents are selected uniformly at random; then, an offspring is generated as described in Section 5.3.1 and 5.3.2. After adding the offspring to the population, we remove one individual with the worst TTP score. Algorithm 16 outlines steps required for $(\mu + 1)$EA.

Algorithm 16 $(\mu + 1)$ Evolutionary Algorithm

```
1: Generate an initial population as explained in Section 5.3.3.
2: while termination criterion is not met do
3:     Generate an offspring and add it to the population $P$.
4:     Discard one individual $p$ from $P$, where $\arg\min_{p \in P} z(p)$.
```

5.3.6 Entropy-based Evolutionary Algorithm

For the sake of comparison, we also propose another algorithm equipped with a structural diversity mechanism to avoid premature convergence. We use a similar mechanism that has been tailored toward EAX and the TSP in [72]. The mechanism employs an entropy-based diversity measure. We name the algorithm entropy-based evolutionary algorithm (EnBEA). Algorithm 17 outlines the steps required for EnBEA.

$$H_e(P) = \sum_{e \in E(P)} h(e) \text{ with } h(e) = -\left(\frac{f(e)}{\sum_{e \in E(P)} f(e)}\right) \cdot \ln\left(\frac{f(e)}{\sum_{e \in E(P)} f(e)}\right)$$

Let $E(P)$ be the set of edges that have been used in the individuals of P, let $h(e)$ be the contribution of edge e to the overall entropy $H(P)$, and let $f(e)$ be the number of individuals in P which use edge e. During the process of EAX, we derive λ AB-cycles from the two parents. Then, we use the AB-cycles one by one to generate λ new tours. Let us denote the first parent and offspring individual by t_p and t_o, respectively. Having computed a packing list for the new tours using the KP operators, we discard the offspring individuals having a TTP score higher than the first parent ($\Delta z = z(t_o) - z(t_p) > 0$). We calculate $\Delta H = H(P \backslash \{t_p\} \cup \{t_o\}) - H(P)$ for the remaining offspring individuals; $H(P \setminus \{t_p\} \cup \{t_o\})$ indicates the population where the first parent is replaced with the offspring individual t_o. If offspring individuals with a positive value of ΔH exist, we replace the parent with the one with the highest TTP score. However, if all offspring individuals result in a less diverse population (all ΔH are negative), we calculate $\frac{\Delta z}{\Delta H}$. Then, we replace the parent with the individual with the smallest $\frac{\Delta z}{\Delta H}$ value. Note that it is not always possible to derive λ AB-cycles

No.	Original Name	No.	Original Name
1	eil51_n50_bounded-strongly-corr_01	18	a280_n279_uncorr_01
2	eil51_n150_bounded-strongly-corr_01	19	rat575_n574_bounded-strongly-corr_01
3	eil51_n250_bounded-strongly-corr_01	20	rat575_n574_uncorr-similar-weights_01
4	eil51_n50_uncorr-similar-weights_01	21	rat575_n574_uncorr_01
5	eil51_n150_uncorr-similar-weights_01	22	dsj1000_n999_bounded-strongly-corr_02
6	eil51_n250_uncorr-similar-weights_01	23	dsj1000_n999_uncorr-similar-weights_06
7	eil51_n50_uncorr_01	24	dsj1000_n999_uncorr_04
8	eil51_n150_uncorr_01	25	u2152_n2151_bounded-strongly-corr_01
9	eil51_n250_uncorr_01	26	u2152_n2151_uncorr-similar-weights_01
10	pr152_n151_bounded-strongly-corr_01	27	u2152_n2151_uncorr_01
11	pr152_n453_bounded-strongly-corr_01	28	fnl4461_n4460_bounded-strongly-corr_01
12	pr152_n151_uncorr-similar-weights_01	29	fnl4461_n4460_uncorr-similar-weights_01
13	pr152_n453_uncorr-similar-weights_01	30	fnl4461_n4460_uncorr_01
14	pr152_n151_uncorr_01	31	dsj1000_n999_uncorr_02
15	pr152_n453_uncorr_01	32	dsj1000_n999_uncorr_03
16	a280_n279_bounded-strongly-corr_01	33	dsj1000_n999_uncorr-similar-weights_03
17	a280_n279_uncorr-similar-weights_01	34	dsj1000_n999_uncorr-similar-weights_04

TABLE 5.1. The names of the TTP instances are used in this chapter.

for the parents. That means the number of AB-cycles and the offspring individuals are capped at λ. In this study, we set λ to 30, which is the same value as used in [72].

Algorithm 17 Entropy-based Evolutionary Algorithm

```
1: Generate an initial population as explained in Section 5.3.3.
2: while termination criterion is not met do
3:    Generate $\lambda$ offspring individuals from $t_{p_1}$ and $t_{p_2}$ using EAX and a KP operator
      and store them in $P'$.
4:    Discard offspring individuals with TTP score less or equal to $t_{p_1}$.
5:    if $P' \neq \emptyset$ then
6:       Calculate $\Delta H = H(P \setminus \{t_p\} \cup \{t_o\}) - H(P)$ for each individuals.
7:       if There exists individuals with $\Delta H \geq 0$ then
8:          Discard the individuals with $\Delta H < 0$.
9:          Replace $t_{p_1}$ with $\arg\max_{t_o \in P'} z(t_o)$.
10:      else
11:         Replace $t_{p_1}$ with $\arg\min_{t_o \in P'} \frac{\Delta z}{\Delta H}$.
```

5.4 Experimental Investigation

In this section, we use the BMBEA to compute a set of solutions for several TTP instances; then, we plot the map to illuminate the distribution of the solutions over the space of f and g. Moreover, we comprehensively compare different search operators and their effects on the distributions and the final maps. We consider the EAX and the 2-OPT for generating tours and the DP and the $(1+1)$EA for computing the packing lists. Employing the operators alternatively, we have four different operator settings. The algorithms are terminated when they reach either of 10000 iterations or 72 hours of CPU time. Here, the iteration is referred to as the main loop of the BMBEA and does not include the time required to obtain f^*, g^* and the initialisation. We use the TTP instances developed in [93]. Table 5.1 presents the names of the instances used in the paper. Please note that we select the first instance of each sub-group except for the dsj1000 where the renting rate is 0, and therefore, these instances constitute

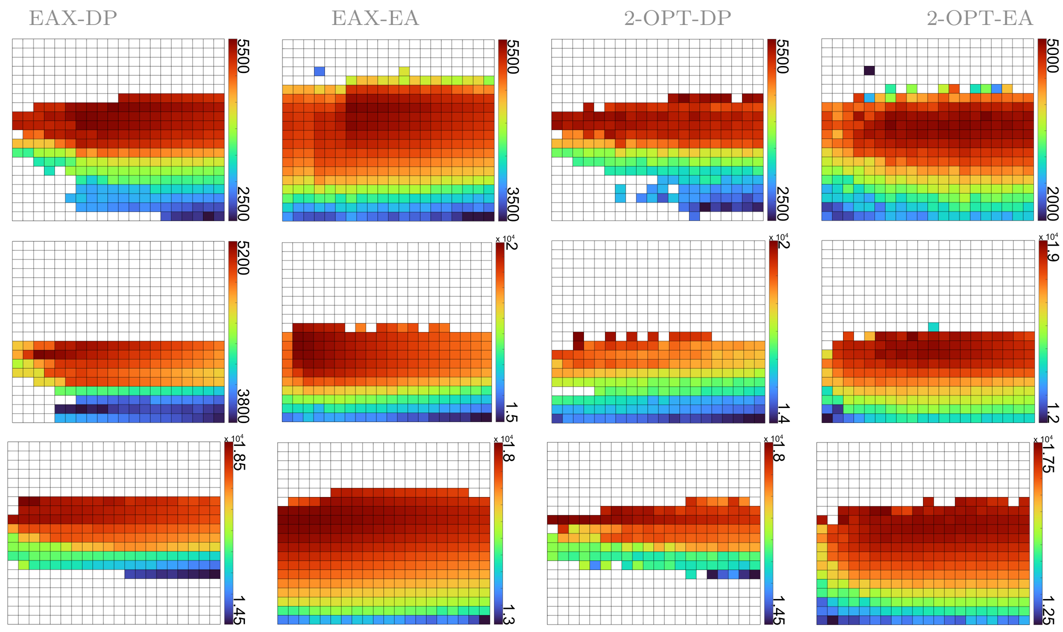


FIGURE 5.2. The distribution of TTP solutions of the four competitors over the behaviour space on instance eil51_n250_uncorr-similar-weights_01 (top), pr152_n453_uncorr_01 (middle), and a280_n279_bounded-strongly-corr_01 (bottom). The cells are coloured based on the average TTP scores of the solutions in the cell over 10 independent runs. The colour bar indicates the TTP scores associated with the colours.

classical knapsack problems. We separate the instances into two categories, small and medium. Since DP's time efficiency is correlated with the number of items, we select instances where $m \leq 500$ for small instances.

5.4.1 Analysis of the maps

This section visualises and scrutinises the final map obtained from the BMBEA using different search operators, namely EAX, 2-OPT, DP, and $(1+1)$EA. Figure 5.2 visualises the final maps obtained from the four competitors in instances 6, 15, and 16. The TSP value increases when we move in the direction of x axis, while moving in the y axis results in a rise in the KP score. Since the TSP is a minimisation and the KP is a maximisation problem, Cell (1,20) consists of the solution with a BD closest to f^* and g^*. The maps clearly indicate the trade-off between the sub-problems in the tested instances since the best TTP solutions should come from Cell (1,20) if there are no interactions. However, the maps show that this is not the case and that we cannot limit our search to the regions that are extremely close to the optimal values for the sub-problems, and we need to extend the search to other neighbouring regions in order to find high-performing solutions for the TTP problem. The maps' cells are coloured based on the average TTP score of the solutions within the cells over 10 independent runs; the hotter the colour, the higher the TTP score. We can observe that the western part of the maps tends to contain better TTP solutions. In 8 out

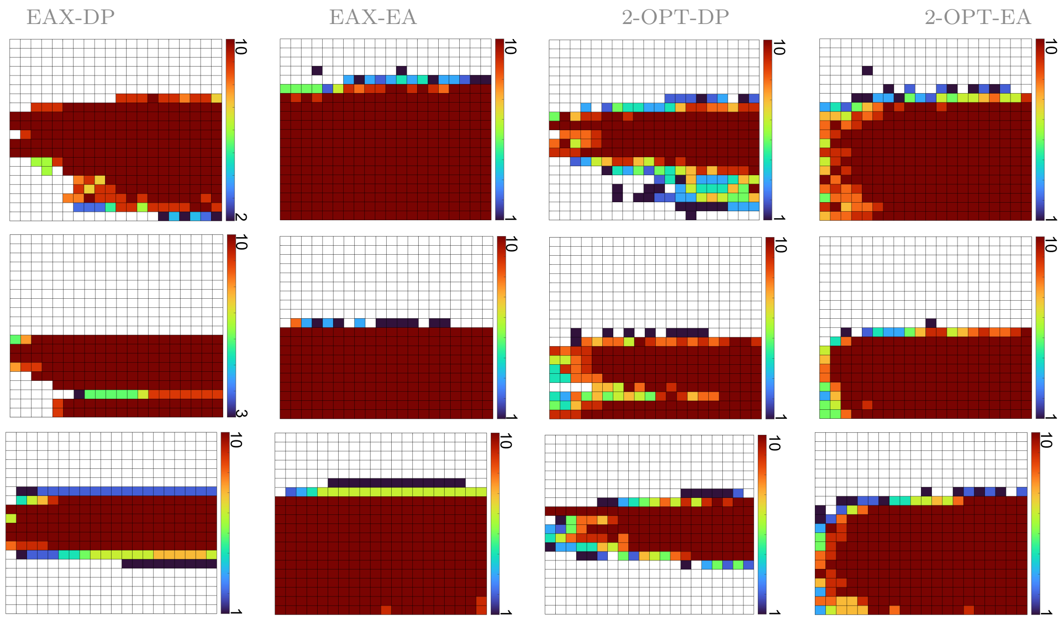


FIGURE 5.3. The frequency of cells housing a TTP solution over 10 independent runs on instance eil51_n250_uncorr-similar-weights_01 (top), pr152_n453_uncorr_01 (middle), and a280_n279_bounded-strongly-corr_01 (bottom). The cells are coloured based on The frequency of cells having a TTP solution in the cell over ten independent runs, as the colour bar indicates the frequencies associated with colours.

of 9 cases, the best solutions are located in a BD of $(1, 1.005)f^*$ and $(0.9, 0.95)g^*$. Moreover, the figure depicts that the maps obtained from BMBEAs using EAX have more hot-coloured cells than the ones with 2-OPT, which shows the consistency of EAX in generating high-quality solutions. Turning to the comparison between DP and $(1 + 1)$EA, the latter can populate a larger part of the map.

Figure 5.3 illustrates the frequency of cells containing a solution over ten independent runs. The instances are the same as Figure 5.2. Here, a hotter colour indicates a higher frequency. The figure depicts that the algorithms cannot populate the cells close to the optimal KP. Because the algorithms compute the packing list as the second level of a bi-level optimisation procedure. Thus, the KP values are constrained by the given tour. Interestingly, most cells corresponding to the KP values close to $(1-\alpha_2)g^*$ also remain empty for the same reason, especially when DP is used. Moreover, one may notice that the most red-coloured cells in Figure 5.2 are coloured red here as well. It illustrates a proportional relationship between the quality of solutions and the frequency. Furthermore, the cells associated with low TSP values (left) of maps are more likely to be empty than the other side. As the TSP value increases, so does the number of tours resulting in such a TSP value rise. This results in a more diverse set of tours and, eventually, a more diverse set of packing lists and a broader range of the KP score.

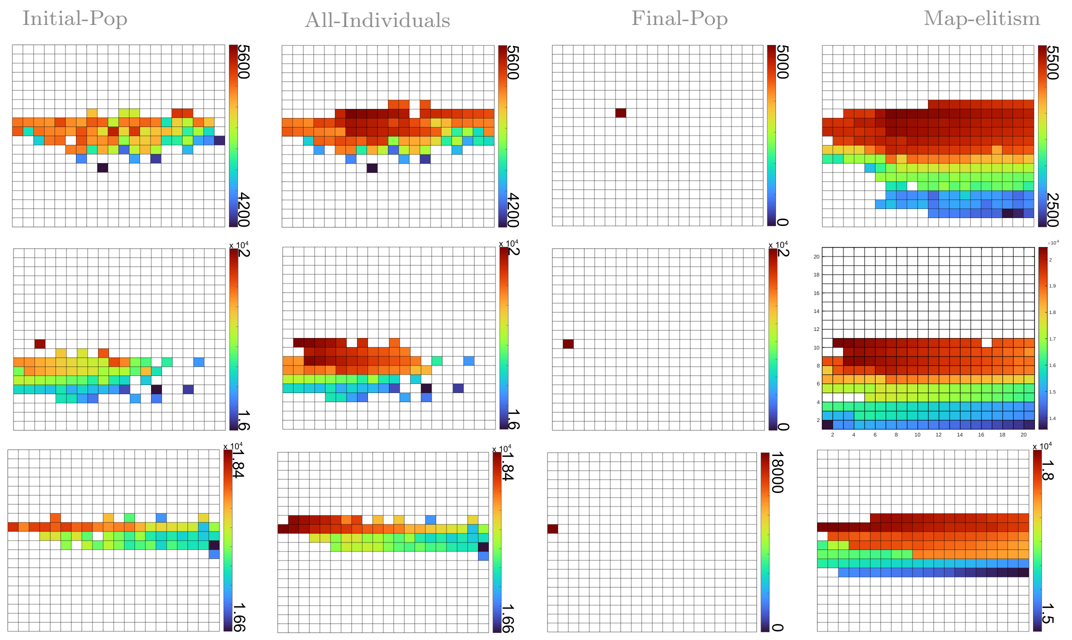


FIGURE 5.4. The illustrations of solutions obtained by $(\mu+1)$EA and Map-elitism in the behavioral space on instance eil51_n250_uncorr-similar-weights_01 (top), pr152_n453_uncorr_01 (middle), and a280_n279_bounded-strongly-corr_01 (bottom). The cells are coloured based on the TTP scores of the solutions in the cell over one single run, as the colour bars show the TTP scores associated with the colours.

MAP-Elitism vs. $(\mu + 1)$EA

In this section, we compare the sets of solutions obtained by BMBEA and $(\mu + 1)$EA. Both algorithms have identical initialisation, parent selection, and offspring generation. Therefore, the difference between these algorithms is limited to survival selection. The aim is to investigate the impact of MAP-Elitism.

Figure 5.4 compares MAP-Elitism and elitism in survival selection. The first three columns belong to $(\mu + 1)$EA and show the initial population, all solutions generated during the search, and the final population, respectively. The fourth column illustrates the population obtained by BMBEA. One can observe that $(\mu+1)$EA converges to a single solution. Since the algorithm uses EAX crossover and DP as operators, it's impossible for the algorithm to find any other solutions from this point. On the other hand, the BMBEA's final population consists of a vast number of solutions with different properties. Thus, it can potentially find better-quality solutions if we let the algorithm continue the search. The other difference we can observe in Figure 5.4 is that the diversity of solutions decreases during the search $(\mu+1)$EA. On the contrary, the diversity increases using MAP-Elitism survival selection.

Maps With The Relaxed Approach

The prefix method can focus on infeasible behavioural space if α_1 and α_2 are set to wrong values. The relaxed method, on the other hand, is more flexible and makes sure that we do not concentrate on an infeasible region of the behavioural space. Figure 5.5 illustrates the α_1 and α_2 in 10 independent runs on instances 1 to 18. Since α_1 and α_2 can differ in each run, we use the figure to compare their distributions in the relaxed method with the values that we used in the prefixed version. Here, after using EAX [72] to generate high-quality tours, we compute an optimal packing list for the tours by DP. Note that we used TSP optimal value as the termination criterion to boost time efficiency and diversity in tours. Figure 5.5 shows α_1 and α_2 belongs to $[0.04 \quad 0.14]$ and $[0.11 \quad 0.33]$, respectively. Since the TSP sub-problem is identical on instances 1 to 9, we can observe a similar trend for α_1 on those instances. This argument is also true for instances 10 to 15 and 16 to 18. On the other hand, KP sub-problems are unique for all cases. Therefore, α_2 is different in each instance. Also, by comparing the distributions of α_2 and the value in the prefixed version (0.2), we can see the values are similar in some cases, such as instances 6, 13, and 18, and different in other cases, such as instances 4 and 7. These observations can support our claim that tuning α_1 and α_2 for each instance separately can yield better results in the prefixed method.

We showed that the BMBEA could bring about diversity in the behavioural space. In the next section, we study the algorithm's performance in terms of TTP score, examine the behavioural diversity's effectiveness in achieving solutions with decent objective value, and compare the introduced algorithm's results with similar selection methods. We expect that using the MAP-Elites-based selection will result in finding better solutions compared to a greedy selection like one the $(\mu+1)$EA uses through enforcing diversity. The diversity should help with premature convergence.

5.4.2 Best found TTP Solutions

In this section, we analyse the performance of BMBEA in solving TTP. First, we investigate the use of the operators and compare the results when using different alternatives of the proposed operators as part of BMBEA. Then, we scrutinise the algorithm's survival selection by comparing BMBEA to $(\mu + 1)$EA. Last, we study the method proposed for relaxing α_1 and α_2 and compared it to BMBEA with fixed α_1 and α_2.

Operators

We now compare the search operators, EAX, 2OPT, DP, and $(1 + 1)$EA, in terms of the best-found TTP solution in this section. We consider instances in a range of 51 to 280 cities and 50 to 453 items from [93]. Table 5.2 shows the average and the best TTP solutions and the average CPU time in ten independent runs for the four competitors

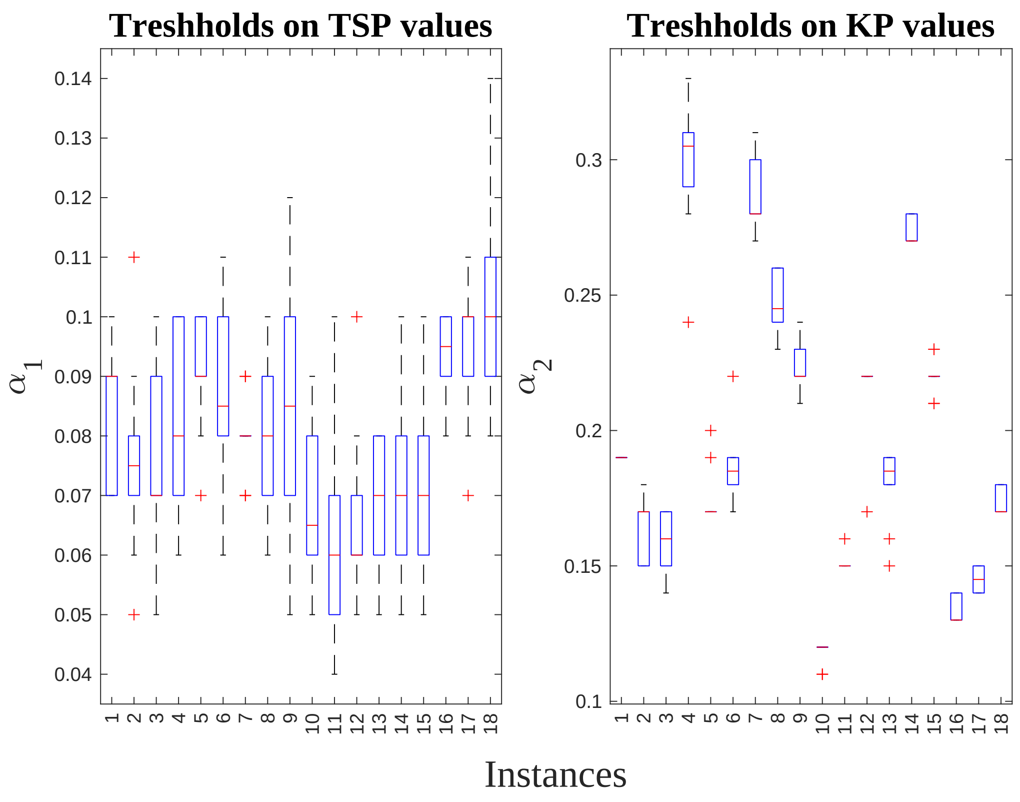


FIGURE 5.5. The values of α_1 and α_2 in the relaxed method

and the best-known TTP values. The best-known values are obtained from [15], and [120]; both these papers compared their results to those of 21 algorithms analysed in [116]. Wagner et al. [116] reported their results on all instances from Table 5.1, while Wuijts and Thierens [120] and Chagas and Wagner [15] used some of the instances in their studies. The best-known values include Chagas and Wagner [15] in instances 1 to 18, 22 to 24, and 28 to 34, and Wuijts and Thierens [120] in instances 1 to 9 and 15 to 18. Note that our termination criterion differs from 10 minutes CPU time in [116], and 2500 local searches in [120]. The results indicate that EAX outperforms 2-OPT in terms of TTP score in most cases. The observations are confirmed by a Kruskal-Wallis test at significance level 5% and the Bonferroni correction. Turning to the comparison of the KP operators, $(1+1)$EA yields very decent objective values and can compete with DP, which results in the optimal packing list. In general, an increase in the size of instances severely affects the run time of the BMBEA using DP. On the other hand, the run times of $(1+1)$EA are significantly shorter. For example, the EAX-EA averagely finishes the 10000 iterations in 240.9 seconds on instance 3. The figure is about 18990.2 seconds for EAX-DP. This is while the algorithm's run time employing $(1+1)$EA remains reasonable. More interestingly, Table 5.2 also indicates that all variants of BMBEA result in high-quality TTP solutions. In instances 10 and 13, the introduced algorithms beat the best TTP scores and can hit the best known on instances 1 and 4.

Since the DP is not time-efficient in larger instances, we consider the $(1+1)$EA for computing the packing list. Table 5.3 shows the results on 12 instances from 575 to 4661 cities and 574 to 4460 items. As one can observe, EAX dominates 2-OPT in these instances. Moreover, the algorithm using EAX improved the best-found solution in 8 out of 12 cases. For example, the TTP score significantly increased from 893 to 1137.5 in instance 22. One can notice that the TTP score of the algorithm using 2-OPT is negative in this instance. Polyakovskiy et al. [93] balanced the instances in the TSP and the KP, but the TSP sub-problem is more dominating in some of the dsj1000 sub-group. The travelling cost is high in these particular instances, and the items do not compensate for the high cost. Having the TSP sub-problem more dominating, it is not surprising that the EAX outperforms the 2-OPT. Moreover, the domination is even stronger in four other instances of the dsj1000 sub-group in a way that the best-known values are negative. We investigate the 4 instances separated from the others due to the dominance of the TSP sub-problem over the KP. It means that the high-quality TTP solutions are closer to the TSP optimal value and farther away from the KP optimal values. The current α_1 and α_2 are set for the balanced instances. Thus, we need to reset α_1 and α_2 to populate the map. Based on initial experimental investigations on these instances, we set α_1 and α_2 to 0.2 and 0.6, respectively. Table 5.4 summarises the results on the 4 instances. The EAX, as expected, outperforms the 2-OPT in all four cases. More importantly, the EAX-based algorithm improved the TTP values for instances 31 and 32 by 1.1 and 37.1, respectively [1]. Note that the results presented in Tables 5.2, 5.3, and 5.4 are different from the ones presented in the conference version [88], which are incorrect due to an implementation mistake.

$(\mu + 1)$**EA vs. BMBEA vs. EnBEA**

We now compare the BMBEA to $(\mu + 1)$EA and EnBEA in solving the problem instances. The difference between these algorithms is in survival selection, where BMBEA selects the next generation based on MAP-Elitism, $(\mu + 1)$EA takes the most elite solutions, and EnBEA uses an entropy-based survival selection. Table 5.5 summarises the results of these algorithms where EAX and DP are considered as the operators. The table shows that the BMBEA has the highest average in 7 cases out of 18, while the figure is 3 and 6 for $(\mu+1)$EA and EnBEA, respectively. The statistical test confirms the BEMBEA outperforms the $(\mu + 1)$EA in cases 13 and 16. It also outperforms EnBEA in instances 15, 17 and 18. Comparing $(\mu + 1)$EA and EnBEA statistically, the first algorithm performs better in the last 3 instances while the latter works better on instance 5. As one can observe, the EnBEA deteriorates in the last 3 instances, which are the largest in the set of instances we used for this comparison. One reason could be that as the size of the instance increases, more AB-cycles can be generated, which affects the convergence pace of EnBEA, and the algorithm requires more runs to converge. The next series of experiments shed light on the matter.

[1]The TTP solutions can be accessed at `https://github.com/NikfarjamAdel/Traveling-Thief-Problem`

Figure 5.4 shows that the $(\mu + 1)$EA converges to a single solution; therefore, there is little hope of finding better solutions by increasing the number of iterations. On the other hand, there is a good chance that an increase in the number of iterations results in better solutions for BMBEA and EnBEA. We investigate this by comparing the results in 10000 and 100000 iterations. Since DP can be time-consuming, we used $(1 + 1)$EA as the KP operator for this round of experiments.

Table 5.6 indicates results for BMBEA, $(\mu+1)$EA, EnBEA when the KP operator is altered to the $(1+1)$EA, and the number of iterations is equal to 10^4 and 10^5. The table shows that BMBEA, $(\mu + 1)EA$, and EnBEA bring about the highest mean of TTP score in 11, 7, and 1 instances, respectively, when the termination criterion is set to 10^4 iterations. Comparing the algorithms statistically, EnBEA is outperformed by the other algorithms. One can notice the differences between this part of the table and Table 5.5. This is because the algorithms need more iterations to converge when we use $(1 + 1)$EA as the KP operator instead of DP. Increasing the number of iterations results in different conclusions. The other part of Table 5.6 can delineate the algorithm's performance.

We can see that both BMBEA and EnBEA statistically outperform the $(\mu + 1)$ in most instances when the number of iterations is set to 10^5. BMBEA, $(\mu+1)$EA and EnBEA have the highest average TTP scores in 8, 2, and 10 instances, respectively. Increasing the number of iterations has no effect on the results of the $(\mu + 1)$EA since the algorithm converges within 10^4 iterations in those instances. On the other hand, it considerably improves the BMBEA and EnBEA performances, while the most significant improvements belong to EnBEA. This shows that both behavioural diversity and structural diversity can be beneficial to prevent premature convergence and escape local optima. Here, we can observe that BMBEA can converge faster than EnBEA and can combine efficiency and effectiveness.

Relaxed Method

Tuning α_1 and α_2 requires a lot of computational effort. It should be done for each instance separately to make sure the area of focus in the feature space is not infeasible and is promising. As we observed in the previous section, the initial values ($\alpha_1 = 0.05$, and $\alpha_2 = 0.2$) result in an infeasible focused area for instances 31 to 34. For this purpose, we proposed the relaxed method where α_1 and α_2 are set based on the initial population , where $\alpha_1 = \left(\frac{f_{max}}{f^*}\right) - 1$ and $\alpha_2 = 1 - \left(\frac{g_{min}}{g^*}\right)$. Here, we investigate the impact of this method on the performance of BMBEA. To do so, we compare the BMBEA using the relaxed method with the prefixed α_1 and α_2.

Table 5.7 summarises the experimental investigation. EAX and $(1 + 1)$EA are considered for the operators. When the termination criterion is set to 10^4 iterations, the prefixed method performs statistically better in 4 out of 27 cases. A meaningful difference can be found in one instance in favour of the relaxed method. At the same time, there is no statistically significant difference in the rest of the cases. However, the average TTP score of the relaxed method is higher in four instances. We can observe that this method performs better in experiments with longer runs. It has a higher average TTP score in 19 cases, while the prefixed approach outperforms it in 5 instances. Both algorithms perform equally in three cases. Statistical tests confirm significant differences in the 2 instances for the relaxed approach. We can conclude that the relaxed method can result in a decent TTP score. At the same time, it eliminates the need to tune two influential parameters. It is noteworthy that both algorithms can beat the best-known TTP values in instances 20 and 25 in the longer runs.

5.5 Conclusion

In this chapter, we incorporated the concept of QD into solving the TTP. To the best of our knowledge, this is the first time the QD concept has been used to solve a combinatorial problem. The behaviour descriptor for our approach is defined on the TSP and the KP scores of a TTP solution. Having described a 2D MAP-Elite-based survival selection, we introduced the BMBEA algorithm to generate high-quality TTP solutions. BMBEA involves EAX crossover to create new tours. Afterwards, the algorithm computes a high-quality packing list by dynamic programming or the (1+1) EA. By visualising the map obtained from BMBEA, we observed the distribution of high-performing TTP solutions over the behavioural space of TSP and KP. Moreover, we conducted a comprehensive experimental comparison involving four different search operators for BMBEA. Moreover, we investigated the impact of MAP-Elitism on the final solutions by comparing it to a simple $(\mu + 1)$EA. The results indicated that MAP-elitism boosts both the diversity and quality of solutions.

It would be interesting to incorporate more complex MAP-Elite approaches such as CVT-MAP-Elites [113] into the introduced algorithm. Using such an approach can discretise the behavioural space more intelligently. Also, it would be beneficial to study the interdependency of the sub-problems theoretically. Moreover, several multi-component combinatorial optimisation problems can be found in literature where QD is highly beneficial to understanding the inter-dependencies of components and the distribution of solutions in the behavioural space.

TABLE 5.2. Comparison of the search operators in terms of the TTP score and CPU time on the small size instances. In columns Stat, the notation X^+ means the median of the measure is better than the one for variant X, X^- means it is worse, and X^* indicates no significant difference. Stat shows the results of the Kruskal-Wallis statistical test at significance level 5% and the Bonferroni correction. The CPU time unit is second.

In.	EAX-DP	(1)			EAX-EA	(2)			Best-known
	Average	Stat	Best	CPU time	Average	Stat	Best	CPU time	value
1	4267.1	$2^*3^+4^+$	4269.4	86.5	4243.1	$1^*3^*4^+$	4269.4	29.4	4269.4
2	7236	$2^*3^*4^+$	7252.8	2275	7086.6	$1^*3^*4^+$	7216.4	112.6	7532
3	11713	$2^*3^+4^+$	11733.9	18990.2	11621.1	$1^*3^*4^+$	11647.5	240.9	12804
4	1449.8	$2^*3^+4^+$	1460	31.1	1443.6	$1^*3^+4^+$	1460	26.9	1460
5	4250.1	$2^*3^+4^+$	4269.6	259.2	4235.1	$1^*3^*4^+$	4248.3	112.2	4365
6	5739.1	$2^*3^+4^+$	5792.2	884.3	5712.1	$1^*3^*4^+$	5792.2	219.7	6359
7	2808	$2^*3^+4^+$	2854.5	40.2	2781.2	$1^*3^*4^+$	2848.1	30.5	2871.1
8	6838.8	$2^*3^+4^+$	6884.4	456	6834.8	$1^*3^*4^+$	6884.4	98.7	7037
9	11753.2	$2^*3^*4^+$	11753.2	2185	11748.2	$1^*3^+4^+$	11753.2	214.7	12478
10	11140.5	$2^+3^+4^+$	11140.5	3639.7	11113.3	$1^-3^*4^+$	11134.8	119.7	11117.4
11	25507.5	$2^*3^+4^+$	25525.5	224858.5	25148.2	$1^*3^+4^*$	25405.3	552.8	25664.4
12	3540.2	$2^*3^*4^+$	3669	253.6	3520.6	$1^*3^*4^+$	3665.4	112.8	3791.9
13	13374.8	$2^*3^*4^+$	13628.3	5234.4	13210.6	$1^*3^*4^+$	13345.5	521.9	13556.9
14	5398.3	$2^*3^*4^+$	5398.3	386.8	5398.3	$1^*3^+4^+$	5398.3	99.5	5615
15	20456.8	$2^*3^+4^+$	20456.8	16047.8	20455.4	$1^*3^+4^+$	20456.8	476.2	20705.8
16	18449.6	$2^+3^*4^+$	18595.5	39024.7	18190.3	$1^-3^*4^*$	18244.5	254.5	19499
17	9163.3	$2^*3^+4^+$	9201.1	1496.5	9122	$1^*3^*4^+$	9176.8	241.9	9998
18	19419.2	$2^*3^+4^+$	19493.4	3588.3	19375.8	$1^*3^+4^+$	19495.4	215.8	20491
In.	2-OPT-DP	(3)			2-OPT-EA	(4)			Best-known
	Average	Stat	Best	CPU time	Average	Stat	Best	CPU time	value
1	4208.3	$1^-2^*4^*$	4237.2	70.5	3969	$1^-2^-3^*$	4104.4	23.2	4296.4
2	7076	$1^*2^*4^+$	7252.8	2049.4	6510.7	$1^-2^-3^-$	6710.9	110.4	7532
3	11550.7	$1^-2^*4^*$	11647.5	16785	10516.4	$1^-2^-3^*$	10964.3	234	12804
4	1407	$1^-2^-4^*$	1428.6	20.1	1364.7	$1^-2^-3^*$	1409.7	21.5	1460
5	4185.9	$1^-2^*4^*$	4255.8	232	3903.8	$1^-2^-3^*$	4018.1	100.5	4365
6	5610.2	$1^-2^*4^*$	5699.2	797.9	5118.4	$1^-2^-3^*$	5277.1	219.1	6359
7	2663.1	$1^-2^*4^*$	2734.3	26.7	2550.6	$1^-2^-3^*$	2619.2	22.3	2871.1
8	6683	$1^-2^*4^*$	6858.8	375.4	6442	$1^-2^-3^*$	6688.4	97.6	7037
9	11480.3	$1^*2^-4^*$	11744.4	1956.5	10890.1	$1^-2^-3^*$	11157.8	205	12478
10	11124.4	$1^-2^*4^+$	11140.1	3070.9	10515.7	$1^-2^-3^-$	10623.3	98.4	11117.4
11	25138	$1^*2^*4^+$	25572.4	116932.7	22722.5	$1^-2^-3^-$	23539.2	553	25664.4
12	3627.5	$1^*2^*4^+$	3752.2	215.5	3266.5	$1^-2^-3^-$	3482.6	94.7	3791.9
13	13190.5	$1^*2^*4^+$	13560.9	4494.9	12407.3	$1^-2^-3^-$	12828.1	477.3	13556.9
14	5238.5	$1^*2^-4^*$	5397.9	321.9	5019.5	$1^-2^-3^*$	5319.3	86.3	5615
15	19725.3	$1^-2^-4^*$	20259.1	12978.6	18931.5	$1^-2^-3^*$	19764.6	474.9	20705.8
16	18273.2	$1^*2^*4^+$	18355.6	36393	17157.6	$1^-2^*3^-$	17307.9	244.2	19499
17	8972.7	$1^-2^*4^*$	9104.9	1479.1	8523.3	$1^-2^-3^*$	8714.4	217.9	9998
18	18901.1	$1^-2^-4^*$	19107.2	3370	18389.3	$1^-2^-3^*$	18763.1	209.9	20491

Table 5.3. Performance of the MAP-Elites-based approach in terms of the TTP score. The notations are in line with Table 5.2.

In.	EAX-EA	(1)			2-OPT-EA	(2)			Best-known
	Average	Stat	Best	CPU time	Average	Stat	Best	CPU time	value
19	32625.3	2^+	33092	835	29687.5	1^-	30065.8	728	32993.1
20	18975.9	2^+	19188.4	708	17622.2	1^-	17803.8	685	19379.7
21	35175.8	2^+	35512.2	696	33456.6	1^-	34455.2	674	35015.2
22	642.3	2^+	1137.5	1625	-4337	1^-	-2693.5	1696	893.4
23	51988.6	2^+	52651.8	1624	48382.8	1^-	49830.4	1685	51303.4
24	29201.5	2^+	32072.9	1627	25214.6	1^-	25618	1620	28304
25	104549.9	2^+	105434.5	7937	95300.4	1^-	96468.9	7975	105908.1
26	71829.9	2^+	73152.8	6914	67954.6	1^-	69060.6	7285	72308.7
27	107975.3	2^+	109395.1	6848	104852.4	1^-	106735	7091	108236.1
28	258901.5	2^+	260839.7	38669	238212.3	1^-	240916	45390	263040.2
29	129168.4	2^+	131072	36670	122606.8	1^-	123626.8	39520	131486.2
30	230888.9	2^+	237097.6	32136	225694.2	1^-	227466.4	31796	233343

Table 5.4. Performance of the MAP-Elites-based approach on the unbalanced instances. The notations are in line with Table 5.2

In.	EAX-EA	(1)			2-OPT-EA	(2)			Best-known
	Average	Stat	Best	CPU time	Average	Stat	Best	CPU time	value
31	-49282	2^+	-48622.8	1555	-51704.2	1^-	-51635	1671	-49149.9
32	-7241	2^+	-4855	1549	-10493	1^-	-9906.8	1709	-7714.6
33	-63137	2^+	-61797.3	1560	–66602	1^-	-63939	1709	-61709.1
34	-24508	2^+	-24263.5	1631	-27573	1^-	-26654	1684	-19215.2

Table 5.5. The comparison of $(\mu+1)EA$ and MAP-Elites in solving TTP using DP as KP operator. The notations are in line with Table 5.2.

In.	Map-Elitism	(1)		$(\mu+1)$	(2)		EnBEA	(3)	
	Average	Stat	Best	Average	Stat	Best	Average	Stat	Best
1	4261.6	2^*3^*	4269.4	4264.9	1^*3^*	4269.4	4266.2	1^*2^*	4269.4
2	7214.8	2^*3^*	7252.8	7193.4	1^*3^*	7252.8	7235.4	1^*2^*	7252.8
3	11705.8	2^*3^*	11733.9	11693.5	1^*3^*	11733.9	11719.4	1^*2^*	11733.9
4	1445.1	2^*3^*	1449.6	1441.6	1^*3^*	1460	1444.4	1^*2^*	1460
5	4259.5	2^*3^*	4286.3	4243.8	1^*3^-	4260.7	4263.7	1^*2^+	4286.3
6	5777.4	2^*3^*	5792.2	5761	1^*3^*	5792.2	5786.2	1^*2^*	5792.2
7	2801.8	2^*3^*	2844.7	2785	1^*3^*	2833.6	2798.5	1^*2^*	2844.7
8	6854.8	2^*3^*	6884.4	6830.3	1^*3^*	6884.4	6846.2	1^*2^*	6884.4
9	11753.2	2^*3^*	11753.2	11753.2	1^*3^*	11753.2	11740.8	1^*2^*	11753.2
10	11140.5	2^*3^*	11140.5	11140.5	1^*3^*	11140.5	11140.5	1^*2^*	11140.5
11	25580.1	2^*3^*	26211.7	25520.8	1^*3^*	25525.5	25607	1^*2^*	26211.7
12	3542.9	2^*3^*	3624	3549.4	1^*3^*	3687.1	3546.9	1^*2^*	3563.6
13	13368.2	2^+3^*	13566.5	13243.4	1^-3^*	13345.5	13345.4	1^*2^*	13345.5
14	5406.7	2^*3^*	5482.2	5398.3	1^*3^*	5398.3	5398.3	1^*2^*	5398.3
15	20419.1	2^*3^*	20456.8	20456.8	1^*3^*	20456.8	20456.6	1^*2^*	20456.8
16	18497.9	2^+3^+	18627.6	18418.2	1^-3^+	18444.1	18353.1	1^-2^-	18404.5
17	9213.2	2^*3^+	9294.7	9164.5	1^*3^+	9277.7	9094.3	1^-2^-	9123.4
18	19412.6	2^*3^+	19804.1	19447.7	1^*3^+	19507	19248.9	1^-2^-	19370.7

TABLE 5.6. The comparison of $(\mu+1)EA$ and MAP-Elites in solving TTP with using $(1+1)$EA as the KP operator. The notations are in line with Table 5.2.

In.	10^4 iterations									10^5 iterations								
	BMBEA (1)			$(\mu+1)$ (2)			$(EnBEA)$ (3)			BMBEA (1)			$(\mu+1)$ (2)			$(EnBEA)$ (3)		
	Average	Stat	Best	Average	Stat	Best	Average	Stat	Best	Average	Stat	Best	Average	Stat	Best	Average	Stat	Best
1	4243.1	2^*3^*	4269.4	4188.9	1^*3^*	4269.4	4231.9	1^*2^*	4255	4269.4	2^+3^*	4269.4	4188.9	1^-3^-	4269.4	4269.4	1^*2^+	4269.4
2	7086.6	2^*3^+	7216.4	7067.4	1^*3^+	7231.1	6954.9	1^-2^-	7068.4	7202.5	2^+3^*	7252.8	7067.4	1^-3^-	7231.1	7220.8	1^*2^+	7252.8
3	11621.1	2^*3^+	11647.5	11637.2	1^*3^+	11689.2	11500.3	1^-2^-	11602.2	11651.9	2^*3^*	11733.9	11637.8	1^*3^-	11695.5	11677.2	1^*2^+	11695.
4	1443.6	2^*3^*	1460	1434.2	1^*3^*	1447.5	1441.6	1^*2^*	1448.5	1458.9	2^+3^*	1460	1434.2	1^-3^-	1447.5	1458.7	1^*2^+	1460
5	4235.1	2^*3^+	4248.3	4202.2	1^*3^*	4246.7	4179	1^-2^*	4211.5	4255.2	2^+3^*	4274.5	4202.2	1^-3^-	4246.7	4277.1	1^*2^+	4286.3
6	5712.1	2^*3^+	5792.2	5663.4	1^*3^*	5729.2	5651.7	1^-2^*	5699.2	5776.6	2^+3^*	5792.2	5663.4	1^-3^-	5729.2	5761.5	1^*2^+	5792.2
7	2774	2^*3^*	2848.1	2748.8	1^*3^*	2844.7	2788.1	1^*2^*	2830.2	2804.7	2^*3^*	2854.5	2748.8	1^*3^-	2844.7	2841.7	1^*2^+	2848.1
8	6834.8	2^+3^*	6884.4	6776.9	1^-3^*	6843.4	6793.2	1^*2^*	6807.1	6858.8	2^+3^*	6884.4	6776.9	1^-3^-	6843.4	6884.4	1^*2^+	6884.4
9	11748.2	2^*3^+	11753.2	11613.2	1^*3^*	11753.2	11616.4	1^-2^*	11753.2	11749	2^+3^*	11753.2	11613.2	1^-3^-	11753.2	11753.2	1^*2^+	11753.
10	11113.3	2^*3^+	11134.8	11133.1	1^*3^+	11137.9	10882.4	1^-2^-	11127.9	11137.7	2^*3^+	11140.4	11133.1	1^*3^*	11137.9	11119.6	1^-2^*	11135.
11	25148.2	2^*3^+	25405.3	25225.8	1^*3^+	25484	23843.3	1^-2^-	24251.7	25396.9	2^*3^*	25524.3	25225.8	1^*3^*	25484	25609.4	1^*2^*	26018.
12	3520.6	2^*3^+	3665.4	3431.2	1^*3^*	3547.6	3422.2	1^-2^*	3517.4	3528.3	2^+3^*	3687.1	3431.2	1^-3^-	3547.6	3527.8	1^*2^+	3755.3
13	13210.6	2^*3^+	13345.5	13278.5	1^*3^+	13345.5	12993.9	1^-2^-	13116.1	13326.6	2^*3^*	13345.5	13278.5	1^*3^-	13345.5	13455.7	1^*2^+	13566.
14	5398.3	2^*3^+	5398.3	5398.3	1^*3^+	5398.3	5376.1	1^-2^-	5398.3	5398.3	2^*3^-	5398.3	5398.3	1^*3^-	5398.3	5351.7	1^+2^+	5482.2
15	20455.4	2^*3^+	20456.8	20267.7	1^*3^+	20456.8	20173	1^-2^-	20392.6	20456.8	2^*3^*	20456.8	20267.7	1^*3^-	20456.8	20472.7	1^*2^+	20496.
16	18190.3	2^-3^+	18244.5	18383.8	1^+3^+	18409.3	17803.6	1^-2^-	17913.3	18381	2^*3^+	18438.6	18392.9	1^*3^+	18409.3	18329.9	1^-2^-	18389.
17	9122	2^*3^+	9176.8	9129.5	1^*3^+	9203.1	8867.7	1^-2^-	9003.3	9249.3	2^+3^*	9331.3	9129.5	1^-3^-	9203.1	9193.8	1^*2^+	9224.2
18	19375.8	2^*3^+	19495.4	19359.9	1^*3^+	19575.5	19022.7	1^-2^-	19158.5	19559	2^+3^*	19738.3	19359.9	1^-3^*	19575.5	19489	1^*2^*	19504.

TABLE 5.7. The comparison of the Relaxed method with the prefixed BMBEA in solving TTP using $(1+1)$EA as KP operator. The notations are in line with Table 5.2.

In.	10^4 iterations						10^5 iterations					
	Prefixed (1)			Relaxed (2)			Prefixed (1)			Relaxed (2)		
	Average	Stat	Best	Average	Stat	Best	Average	Stat	Best	Average	Stat	Best
1	4243.1	2^*	4269.4	4246	1^*	4269.4	4269.4	2^*	4269.4	4269.4	1^*	4269.4
2	7086.6	2^*	7216.4	7116.7	1^*	7231.	7202.5	2^*	7252.8	7195.4	1^*	7252.8
3	11621.1	2^*	11647.5	11619.5	1^*	11697.5	11651.9	2^*	11733.9	11667.9	1^*	11733.3
4	1443.6	2^*	1460	1437.1	1^*	1445.8	1458.9	2^*	1460	1458.7	1^*	1460
5	4235.1	2^*	4248.3	4235.8	1^*	4260.	4255.2	2^*	4274.5	4264.2	1^*	4286.3
6	5712.1	2^+	5792.2	5655	1^-	5729.	5776.6	2^*	5792.2	5762.1	1^*	5792.2
7	2781.2	2^*	2848.1	2799.1	1^*	2840	2804.7	2^*	2854.5	2847.5	1^*	2854.5
8	6834.8	2^*	6884.4	6833.8	1^*	6884.4	6858.8	2^-	6884.4	6884.4	1^+	6884.4
9	11748.2	2^*	11753.2	11744.7	1^*	11753.	11749	2^*	11753.2	11753.2	1^*	11753.2
10	11113.3	2^*	11134.8	11103	1^*	11135.4	11137.7	2^*	11140.4	11136.9	1^*	11137.9
11	25148.2	2^*	25405.3	25036.9	1^*	25332.2	25396.9	2^*	25524.3	25441	1^*	25525.5
12	3520.6	2^*	3665.4	3487.4	1^*	3547.6	3528.3	2^*	3687.1	3564.7	1^*	3789.8
13	13210.6	2^*	13345.5	13270	1^*	13345.5	13326.6	2^*	13345.5	13345.5	1^*	13345.5
14	5398.3	2^+	5398.3	5387.6	1^-	5398.3	5398.3	2^*	5398.3	5398.3	1^*	5398.3
15	20455.4	2^*	20456.8	20421.6	1^*	20456.8	20456.8	2^*	20456.8	20456.8	1^*	20456.8
16	18190.3	2^*	18244.5	18187.2	1^*	18330.5	18381	2^*	18438.6	18388.1	1^*	18472
17	9122	2^+	9176.8	9074.2	1^-	9143.1	9249.3	2^*	9331.3	9237.7	1^*	9294.1
18	19375.8	2^*	19495.4	19247	1^*	19432.	19559	2^*	19738.3	19573.8	1^*	19628.9
19	32625.3	2^*	33092	32354.2	1^*	33080	33358.4	2^*	33759	33496.8	1^*	34007
20	18975.9	2^*	19188.4	18965.4	1^*	19092.8	19279.8	2^*	19538.3	19423.5	1^*	19650.7
21	35175.8	2^*	35512.2	35101.1	1^*	35429.5	35720.2	2^*	36021.4	35762.2	1^*	36133.8
22	642.3	2^*	1137.5	541.5	1^*	879.3	1535.5	2^-	2621.1	2356.1	1^+	2964
23	51988.6	2^*	52651.8	51885.5	1^*	52345.7	52292.8	2^*	52973.6	52954.2	1^*	55260.9
24	29201.5	2^*	32072.9	28711.7	1^*	29299.3	29492.5	2^*	32085.1	29500.7	1^*	30411
25	104549.9	2^+	105434.5	103863.4	1^-	105025	105442.9	2^*	106465.7	106146.9	1^*	107250.6
26	71829.9	2^*	73152.8	71615.1	1^*	73141	72110.7	2^*	73348.2	72149.4	1^*	73291.9
27	107975.3	2^-	109395.1	108885.5	1^+	109929	109402.2	2^*	110346	109846	1^*	110681

Chapter 6

Analysis of inter-dependency of the Traveling Thief Problem

6.1 Introduction

In the previous chapter, we introduced a QD-based method to study the TTP. We showed the effectiveness of the introduced QD method in illustrating the distribution of high-quality solutions within the behavioural space of the TSP and the KP. Moreover, we conducted comprehensive experiments showing the Qd-based algorithm results in TTP solutions with very decent objective values. Those findings motivate us to investigate the TTP in the context of EDO. Also, there is a lack of studies focusing on EDO in multi-component combinatorial problems that must be addressed. Several advantages can be found in having a structurally diverse set of high-quality solutions for TTP that EDO brings about. First, we can study the inter-dependency of the sub-problems in terms of structural diversity and find the best method to maximise it. Second, EDO provides us with invaluable insight into the solutions space. For example, it can show which elements of an optimal/near-optimal solution can be replaced and which are irreplaceable. Finally, it brings about robustness against the minor changes as mentioned in Chapter 2.

To the best of our knowledge, this study is the first to investigate EDO in the context of a multi-component problem. We first establish a method to calculate the structural diversity of TTP solutions. Then, we use a similar bi-level EA introduced in the previous chapter, this time to maximise the structural diversity. Similar to the EA presented in the previous chapter, the first level involves generating the TSP part of a TTP solution, whereby the second level is an inner algorithm to optimise the KP part of the solution with respect to the first part. The difference is in survival selection, where an EDO-based selection is exercised to maximise the diversity.

We first examine the impact that incorporating different inner algorithms into the EA can make on the diversity of the solutions. Moreover, We empirically study the inter-dependency between the sub-problems and show how focusing on the diversity of one sub-problem affects the other's and determines the best method to obtain diversity. Interestingly, the results indicate that focusing on overall diversity brings about greater KP diversity than solely emphasising KP diversity. In addition, we compare the set of solutions obtained from the introduced algorithm with a recently developed QD-based EA in terms of structural diversity. The results show that the introduced bi-level EA can bring higher structural diversity for most test instances. We also conduct a simulation test to evaluate the robustness of populations obtained from the two algorithms against changes in the problem.

The chapter also presents a co-evolutionary algorithm (Co-EA) where two populations evolve around the concepts of QD and EDO simultaneously. The co-evolution results in several advantages:

- QD provides researchers with invaluable information about the distribution of best-performing solutions in behavioural space and enables decision-makers to select the best solution having their desirable behaviour. On the other hand, EDO provides us with robustness against imperfect modelling and minor changes in problems. we can benefit from both paradigms by using the Co-EA.
- Optimal or close-to-optimal solutions are required in most EDO studies for initialization. The Co-EA eliminates this restriction.
- We expect the Co-EA to bring about better results, especially in terms of structural diversity, since the previous frameworks are built upon a single solution (the optimal solution). The Co-EA eliminates this drawback.
- The Co-EA benefits from a self-adaptation method to tune and adjust some hyper-parameters during the search improving the results meaningfully.

The work of this chapter is based on a conference paper [87] presented at the genetic and evolutionary computation conference (GECCO 2022). Also, the Co-EA is based on another conference paper [86] presented in the parallel problem solving from nature (PPSN 2022). The remainder of the chapter is structured as follows. We formally define the diversity in a set of TTP solutions in Section 6.2. In Section 6.3, We introduce the two-stage EA. A comprehensive experimental investigation is conducted in Section 6.4. The Co-EA is introduced in Section 6.5. We conduct a comprehensive experimental investigation to evaluate Co-EA in Section 6.6. Finally, we finish with some concluding remarks.

6.2 Diversity in TTP

This chapter aims to compute a structurally diverse set of TTP solutions that all comply with a minimum quality threshold but differ in terms of structural properties. In other words, the objective is to maximise the diversity of the set of solutions subject to a quality constraint. Let denote the set of TTP solutions by $P = \{p_1(x_1, y_1), \cdots, p_\mu(x_\mu, y_\mu)\}$, where $|P| = \mu$, and a TTP solution p consists of a TSP tour x and a packing selection y. Therefore, we can formally formulate the problem as:

$$\begin{aligned}
&MaxH(P) \\
&\text{subject to} \\
&z_p \geq (1-\alpha)z^* && && \forall p \in P \\
&\sum_{j=1}^{m} w_j y_{jp} \leq W && \forall p \in P && y_{jp} \in \{0,1\}
\end{aligned}$$

Where $H(P)$ is a measure quantifying the diversity of P, z^* is the optimal or the best-known value of objective function z for a given TTP instance, α is the acceptable quality threshold, and y_{jp} shows $y_j \in p$. In line with most of the studies in EDO literature, we assumed that the optimal or high-quality solution of TTP instances is already known.

To maximise the diversity, we require a measure to quantify the diversity of a set of solutions. As mentioned, a TTP solution includes two different parts, a tour and a packing list. That means a function is required to calculate the structural diversity of tours and another one for packing lists. We adopt the well-known information-theoretic concept of entropy for this purpose.

We employ the diversity measure based on entropy to compute the entropy of the tours. Let P be a set of TTP solutions. Here, the diversity is defined on the proportion of edges including in $E(P)$, where $E(P)$ is the set of edges included in P. The edge entropy of P can be calculated from:

$$H_e(P) = \sum_{e \in E(P)} h(e) \text{ with } h(e) = -\left(\frac{f(e)}{2n\mu}\right) \ln\left(\frac{f(e)}{2n\mu}\right).$$

where $h(e)$ is the contribution of an edge $e \in E$ to the entropy, and $f(e)$ is the number of tours in P including e. The contribution of edges with zero frequency is equal to zero ($h(e) = 0 \iff f(e) = 0$). $2n\mu$ is the summation of the frequency of all edges over the population.

The same concept is adopted for the calculation of the entropy of items. The

Algorithm 18 Two-stage-EA

```
Require: Population $P$, minimal quality threshold $z_{min}$
1: while termination criterion is not met do
2:     Choose $x_1$ and $x_2 \in P$ uniformly at random, and generate one tour $x_3$ by crossover.
3:     Generate a corresponding packing list $(y_3)$ by a KP operator to have a complete TTP solution
4:     if $z((x_3, y_3))) \geq z_{min}$ then
5:         Add $(x_3, y_3)$ to $P$.
6:     if $|P| = \mu + 1$ then
7:         Remove one individual $(x, y))$ from $P$, where $p = \arg\max_{(x,y) \in P} H(P \setminus \{(x, y)\})$.
```

diversity of the packing list on the proportion of items being included in $P(I)$, where $P(I)$ is the set of items included in P. The item entropy of P can be computed from:

$$H_i(P) = \sum_{i \in P(I)} h(i) \text{ with } h(i) = -\left(\frac{f(i)}{\sum_{i \in I} f(i)}\right) \ln\left(\frac{f(i)}{\sum_{i \in I} f(i)}\right).$$

where $h(i)$ is the contribution of an item $i \in I$ to the entropy, and $f(i)$ is the number of packing lists in P including i. The contribution of items with zero frequency is equal to zero $(h(i) = 0 \iff f(i) = 0)$.

A simple way to calculate the overall entropy is to sum up the entropy of edges and items. This is because H_e and H_i are basically the summation of the contribution of edges and items. Therefore, we have: $H(P) = H_e(P) + H_i(P)$

6.3 Bi-level Evolutionary Algorithm

We introduce a bi-level EA to compute a diverse set of TTP solutions. The EA is started with an initial population that all individuals comply with the quality constraint. The procedure to construct such a population will be explained later. Having selected two tours uniformly at random, the EA generates a new tour by crossover. Then, an inner algorithm is initiated to compute a corresponding packing list for the new tour in order to have a complete TTP solution; we refer to the inner algorithms as the KP operators. If the TTP score of the new solution is higher than the minimum requirement, it will be added to the population; otherwise, it will be discarded. Finally, an individual with the minimum contribution to the diversity of the population will be discarded if the size of the population is $\mu + 1$. These steps are continued until a termination criterion is met. Algorithm 18 outlines the bi-level EA. The TSP and the KP operators are in line with the previous chapter.

Algorithm 19 Initial Population Procedure

```
Require: A TTP solution $(x, y)$ complying the quality criterion, population size $\mu$
1: while $|P| < \mu$ do
2:     Choose $(x, y) \in P$ uniformly at random, generate a tour $x'$ by mutation.
3:     Compute a packing list $y'$ by the DP to form $(x', y')$)
4:     if $z(p') \geq z_{min}$ then
5:         Add $(x', y')$ to $P$.
```

TABLE 6.1. Comparison of the KP operators. In columns Stat, the notation X^+ means the median of the measure is better than the one for variant X, X^- means it is worse, and X^* indicates no significant difference. Stat shows the results of Mann-Whitney U-test at significance level 5%

Int	DP	(1)	EA	(2)	DP	(1)	EA	(2)	DP	(1)	EA	(2)
	H	Stat	H	Stat	H_e	Stat	H_e	Stat	H_i	Stat	H_i	Stat
1	8.5	2^*	8.3	1^*	5.4	2^-	5.7	1^+	3	2^+	2.6	1^-
2	9	2^+	8.8	1^-	5.2	2^+	5	1^-	3.8	2^*	3.8	1^*
3	9.5	2^+	9.3	1^-	5.1	2^+	5	1^-	4.3	2^*	4.4	1^*
4	7.1	2^*	7.2	1^*	5.3	2^+	5.2	1^-	1.9	2^*	2	1^*
5	8.7	2^+	8.3	1^-	5.2	2^+	5	1^-	3.4	2^+	3.3	1^-
6	8.9	2^+	8.8	1^-	5.2	2^+	5	1^-	3.8	2^*	3.8	1^*
7	7.9	2^*	7.8	1^*	5.3	2^*	5.3	1^*	2.5	2^*	2.5	1^*
8	8.6	2^*	8.6	1^*	5	2^*	5	1^*	3.5	2^-	3.6	1^+
9	9.2	2^*	9.2	1^*	5.1	2^*	5	1^*	4.1	2^-	4.1	1^+
10	9.8	2^*	9.8	1^*	6	2^+	5.9	1^-	3.8	2^-	3.9	1^+
11	10.7	2^-	10.7	1^+	6	2^*	6	1^*	4.7	2^*	4.7	1^*
12	8.6	2^-	8.8	1^+	6	2^*	5.9	1^*	2.7	2^-	2.8	1^+
13	9.9	2^*	10	1^*	6	2^+	5.8	1^-	3.9	2^-	4.2	1^+
14	9.4	2^*	9.4	1^*	5.9	2^*	5.9	1^*	3.4	2^*	3.5	1^*
15	10.6	2^*	10.6	1^*	6	2^*	6	1^*	4.6	2^-	4.6	1^+
16	10.7	2^*	10.7	1^*	6.5	2^+	6.4	1^-	4.3	2^-	4.4	1^+
17	10.1	2^*	10.1	1^*	6.5	2^+	6.4	1^-	3.6	2^*	3.8	1^*
18	10.7	2^*	10.7	1^*	6.5	2^+	6.4	1^-	4.2	2^-	4.2	1^+

6.3.1 Initial Population

As mentioned, we assumed that we know the optimal/near-optimal solution for given TTP instances; such an assumption is in line with most studies in the literature of EDO. The procedure is initialised with a single high-quality solution solution (x, y) in P, where $z(x, y) \in ((1 - \alpha)z^* z^*)$. First, an individual $(x, y) \in P$ is selected uniformly at random. Then, the tour of the individual (x) is mutated by 2-OPT, which is a well-known random neighbourhood search in TSP. Afterwards, we compute a packing list y' by KP and match it with the mutated tour x' to have a TTP solution (x', y'). If (x', y') complies with the quality constraint, it will be added to P; otherwise, it is discarded. We continue these steps until $|P| = \mu$. Note that We used the algorithm introduced in [88] to obtain the initial (x, y). Algorithm 19 outlines the initialising procedure.

TABLE 6.2. Comparison of different fitness functions (DP used as the KP operator). Stat shows the results of the Kruskal-Wallis statistical test at significance level 5% and Bonferroni correction. The notations are in line with Table 6.1

Ins	H	(1)	H_e	(2)	H_i	(3)	H	(1)	H_e	(2)	H_i	(3)	H	(1)	H_e	(2)	H_i	(3)
	H	Stat	H	Stat	H	Stat	H_e	Stat	H_e	Stat	H_e	Stat	H_i	Stat	H_i	Stat	H_i	Stat
1	8.5	2^*3^+	8.3	1^*3^+	7.5	1^-2^-	5.4	2^-3^+	5.8	1^+3^+	4.8	1^-2^-	3	2^+3^+	2.6	1^-3^*	2.7	1^-2^*
2	9	2^*3^+	9.1	1^*3^+	8.6	1^-2^-	5.2	2^-3^+	5.4	1^+3^+	4.8	1^-2^-	3.8	2^+3^+	3.7	1^-3^*	3.8	1^-2^*
3	9.5	2^*3^+	9.6	1^*3^+	9.1	1^-2^-	5.1	2^-3^+	5.3	1^+3^+	4.8	1^-2^-	4.3	2^*3^*	4.3	1^*3^*	4.3	1^*2^*
4	7.1	2^*3^+	6.8	1^*3^+	6.4	1^-2^-	5.3	2^*3^+	5.4	1^*3^+	4.8	1^-2^-	1.9	2^+3^*	1.5	1^-3^*	1.6	1^*2^*
5	8.7	2^+3^+	8.3	1^-3^*	7.7	1^-2^*	5.2	2^-3^+	5.4	1^+3^+	4.8	1^-2^-	3.4	2^+3^+	2.8	1^-3^*	2.9	1^-2^*
6	8.9	2^*3^+	8.7	1^*3^+	8.3	1^-2^-	5.2	2^-3^+	5.3	1^+3^+	4.8	1^-2^-	3.8	2^+3^+	3.4	1^-3^*	3.5	1^-2^*
7	7.9	2^*3^+	7.8	1^*3^+	7.3	1^-2^-	5.3	2^*3^+	5.4	1^*3^+	4.8	1^-2^-	2.5	2^+3^+	2.4	1^-3^*	2.5	1^-2^*
8	8.6	2^-3^+	8.7	1^+3^+	8.2	1^-2^-	5	2^-3^+	5.2	1^+3^+	4.7	1^-2^-	3.5	2^+3^+	3.5	1^-3^*	3.5	1^-2^*
9	9.2	2^*3^+	9.3	1^*3^+	8.8	1^-2^-	5.1	2^*3^+	5.2	1^*3^+	4.7	1^-2^-	4.1	2^+3^*	4.1	1^-3^*	4.1	1^*2^*
10	9.8	2^*3^+	9.9	1^*3^+	9.6	1^-2^-	6	2^-3^+	6.2	1^+3^+	5.8	1^-2^-	3.8	2^+3^*	3.7	1^-3^-	3.8	1^*2^+
11	10.7	2^*3^+	10.8	1^*3^+	10.5	1^-2^-	6	2^*3^+	6.1	1^*3^+	5.8	1^-2^-	4.7	2^+3^*	4.7	1^-3^*	4.7	1^*2^*
12	8.6	2^*3^+	8.6	1^*3^+	8.5	1^-2^-	6	2^*3^+	6	1^*3^+	5.8	1^-2^-	2.7	2^+3^*	2.6	1^-3^-	2.7	1^*2^+
13	9.9	2^*3^+	9.9	1^*3^+	9.7	1^-2^-	6	2^*3^+	6.1	1^*3^+	5.8	1^-2^-	3.9	2^+3^*	3.8	1^-3^-	3.9	1^*2^+
14	9.4	2^*3^+	9.4	1^*3^+	9.2	1^-2^-	5.9	2^*3^+	6	1^*3^+	5.8	1^-2^-	3.4	2^+3^*	3.4	1^-3^-	3.4	1^*2^+
15	10.6	2^*3^+	10.6	1^*3^+	10.4	1^-2^-	6	2^*3^+	6	1^*3^+	5.8	1^-2^-	4.6	2^*3^*	4.6	1^*3^*	4.6	1^*2^*
16	10.7	2^*3^+	10.8	1^*3^+	10.6	1^-2^-	6.5	2^*3^+	6.6	1^*3^+	6.4	1^-2^-	4.3	2^+3^*	4.2	1^-3^*	4.2	1^*2^*
17	10.1	2^*3^*	9.9	1^*3^*	9.9	1^*2^*	6.5	2^*3^+	6.6	1^*3^+	6.4	1^-2^-	3.6	2^+3^*	3.4	1^-3^-	3.6	1^*2^+
18	10.7	2^*3^+	10.8	1^*3^+	10.6	1^-2^-	6.5	2^*3^+	6.6	1^*3^+	6.4	1^-2^-	4.2	2^+3^*	4.2	1^-3^-	4.2	1^*2^+

6.4 Experimental Investigation

Here, we conduct a comprehensive experimental investigation on the introduced framework to analyse the inter-dependency of the TTP's sub-problems in terms of structural diversity and find the best method to maximise it. First, we compare the two KP search operators, DP and $(1+1)$EA; then, we incorporate the H, H_e, and H_i into the algorithm as the fitness function and analyse the populations obtained. Finally, we conduct a comparison of the introduced framework with the QD-based EA introduced in the previous chapter in terms of structural diversity and robustness against small changes in the availability of edges and items. In terms of the experimental setting, we used 18 TTP instances from [93], and the algorithms are terminated after 10000 iterations. The TTP instances are in line with the previous chapter. The internal termination criterion for the $(1+1)$EA is set to $2m$ based on preliminary experiments. We consider 10 independent runs for each algorithm on each test instance.

6.4.1 Comparison in KP search operators operators

In this section, we compute two sets of solutions for each test instance, one by use of DP and another with $(1+1)$EA, and scrutinise the diversity of the sets. Here, H serves as fitness function and α and μ are set to 0.1 and 50, respectively. Table 6.1 summarises the results. As Table 6.1 shows, the use of DP results in a population with higher diversity in edges (H_e), while H_i is higher in the population obtained from $(1+1)$EA in most cases. Turning to overall diversity (H), the use of DP brings about populations with higher diversity in 4 out of 18 cases. On the other hand, there

TABLE 6.3. Comparison of different fitness function (EA used as the KP operator).The notations are in line with Table 6.2

Ins	H	(1)	H_e	(2)	H_i	(3)	H	(1)	H_e	(2)	H_i	(3)	H	(1)	H_e	(2)	H_i	(3)
	H	Stat	H	Stat	H	Stat	H_e	Stat	H_e	Stat	H_e	Stat	H_i	Stat	H_i	Stat	H_i	Stat
01	8.3	2^*3^+	8.3	1^*3^+	7.5	1^-2^-	5.7	2^*3^+	5.8	1^*3^+	4.9	1^-2^-	2.6	2^+3^*	2.5	1^-3^-	2.6	1^*2^+
02	8.8	2^*3^+	9	1^*3^+	8.4	1^-2^-	5	2^-3^+	5.3	1^+3^+	4.7	1^-2^-	3.8	2^+3^*	3.7	1^-3^-	3.8	1^*2^+
03	9.3	2^*3^+	9.4	1^*3^+	9	1^-2^-	5	2^-3^+	5.2	1^+3^+	4.7	1^-2^-	4.4	2^+3^*	4.2	1^-3^-	4.3	1^*2^+
04	7.2	2^+3^+	6.9	1^-3^+	6.4	1^-2^-	5.2	2^-3^+	5.4	1^+3^+	4.8	1^-2^-	2	2^+3^+	1.5	1^-3^*	1.6	1^-2^*
05	8.3	2^*3^+	8.1	1^*3^+	7.7	1^-2^-	5	2^-3^*	5.4	1^+3^+	4.7	1^*2^-	3.3	2^+3^+	2.7	1^-3^*	3	1^-2^*
06	8.8	2^+3^+	8.6	1^-3^*	8.2	1^-2^*	5	2^-3^+	5.3	1^+3^+	4.7	1^-2^-	3.8	2^+3^*	3.3	1^-3^-	3.6	1^*2^+
07	7.8	2^*3^+	7.8	1^*3^+	7.3	1^-2^-	5.3	2^*3^+	5.4	1^*3^+	4.8	1^-2^-	2.5	2^+3^*	2.4	1^-3^*	2.5	1^*2^*
08	8.6	2^*3^+	8.7	1^*3^+	8.3	1^-2^-	5	2^-3^+	5.2	1^+3^+	4.7	1^-2^-	3.6	2^+3^*	3.5	1^-3^-	3.6	1^*2^+
09	9.2	2^*3^+	9.3	1^*3^+	8.8	1^-2^-	5	2^-3^+	5.2	1^+3^+	4.7	1^-2^-	4.1	2^+3^*	4.1	1^-3^-	4.1	1^*2^+
10	9.8	2^*3^+	10	1^*3^+	9.6	1^-2^-	5.9	2^-3^+	6.2	1^+3^+	5.7	1^-2^-	3.9	2^+3^*	3.8	1^-3^-	3.9	1^*2^+
11	10.7	2^*3^+	10.8	1^*3^+	10.5	1^-2^-	6	2^*3^+	6.1	1^*3^+	5.7	1^-2^-	4.7	2^*3^*	4.7	1^*3^-	4.8	1^*2^+
12	8.8	2^*3^*	8.7	1^*3^*	8.7	1^*2^*	5.9	2^*3^+	6	1^*3^+	5.7	1^-2^-	2.8	2^+3^*	2.6	1^-3^-	2.9	1^*2^+
13	10	2^+3^*	9.9	1^-3^*	9.9	1^*2^*	5.8	2^-3^+	6.1	1^+3^+	5.7	1^-2^-	4.2	2^+3^*	3.8	1^-3^-	4.2	1^*2^+
14	9.4	2^*3^+	9.4	1^*3^+	9.2	1^-2^-	5.9	2^-3^+	6	1^+3^+	5.8	1^-2^-	3.5	2^+3^*	3.4	1^-3^-	3.5	1^*2^+
15	10.6	2^*3^+	10.6	1^*3^+	10.3	1^-2^-	6	2^*3^+	6	1^*3^+	5.7	1^-2^-	4.6	2^+3^*	4.6	1^-3^*	4.6	1^*2^*
16	10.7	2^*3^*	10.8	1^*3^+	10.7	1^*2^-	6.4	2^-3^+	6.6	1^+3^+	6.3	1^-2^-	4.4	2^+3^*	4.2	1^-3^-	4.4	1^*2^+
17	10.1	2^+3^*	9.9	1^-3^-	10.1	1^*2^+	6.4	2^-3^*	6.6	1^+3^+	6.3	1^*2^-	3.8	2^+3^*	3.3	1^-3^-	3.7	1^*2^+
18	10.7	2^*3^*	10.7	1^*3^+	10.6	1^*2^-	6.4	2^*3^+	6.6	1^*3^+	6.3	1^-2^-	4.2	2^+3^*	4.2	1^-3^-	4.2	1^*2^+

are 2 cases in which $(1+1)$EA outperforms the DP. There are found no significant differences in overall diversity for the rest of the test instances. One may ask the question why using DP results in a higher H_e, while EAX is used to generate new tours in both competitors. One explanation is that DP computes the same packing list for two identical tours. This is while $(1+1)$EA can generate different packing lists, which results in a higher H_i but a lower H_e.

6.4.2 Comparison in fitness functions

Next, we investigate the use of H_e or H_i as the fitness functions instead of H. Table 6.2 compares three algorithms using the fitness functions where DP is used as the KP operator. The table shows there are no significant differences in overall diversity when either H or H_e serve as the fitness function. However, the EA using item diversity (H_i) results in the populations with an overall entropy significantly less than the other EAs. It also gets outperformed by the EA using H in terms of item diversity. This is because the introduced framework is a bi-level optimisation procedure where it generates a tour first; then, it computes the packing list based on the tour. Therefore, the use of diversity in edge can aid in increasing the diversity of items, especially where the EA uses DP. Figure 6.1 depicts the trajectories of the EAs using the three fitness functions over 10000 iterations in the test instances 1 and 16. The figure explicitly confirms the previous observations; using H_i as the fitness function makes the EA incapable of maximising overall and edge diversity. It also gets outperformed in terms of item entropy H_i. On the other hand, incorporating H_e as the fitness functions results in decent overall and edge diversity. However, it can not increase the item entropy. Figure 6.1 also shows the EAs using H and H_e as the fitness function do not

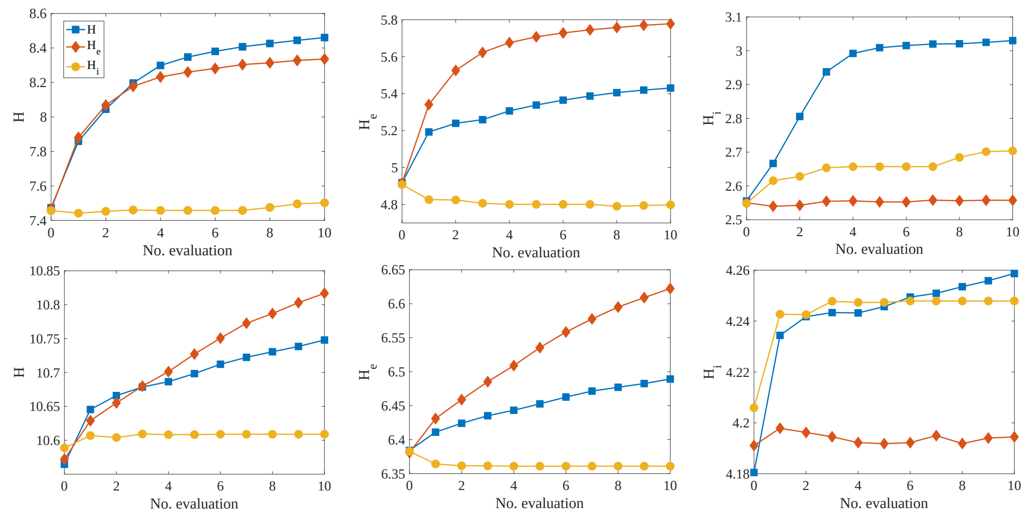


FIGURE 6.1. Representation of trajectories of incorporation of the different fitness functions, H, H_e, and H_i over test instance 1 (first row), and test instance 16 (second row)

TABLE 6.4. Comparison of the EDO and QD (DP used as the KP operator). The notations are in line with Table 6.1.

Int	EDO (1)		QD (2)		EDO (1)		QD (2)		EDO (1)		QD (2)	
	H	Stat	H	Stat	H_e	Stat	H_e	Stat	H_i	Stat	H_i	Stat
01	9.1	2^+	8.1	1^-	5.9	2^+	5.1	1^-	3.2	2^+	3	1^-
02	9.8	2^+	9.1	1^-	5.6	2^+	5.1	1^-	4.2	2^+	3.9	1^-
03	10.2	2^+	9.5	1^-	5.5	2^+	5.1	1^-	4.6	2^+	4.4	1^-
04	9.1	2^+	7.7	1^-	5.9	2^+	5.2	1^-	3.2	2^+	2.5	1^-
05	9.7	2^+	8.7	1^-	5.7	2^+	5.1	1^-	4	2^+	3.5	1^-
06	10.3	2^+	9.2	1^-	5.8	2^+	5.1	1^-	4.5	2^+	4.1	1^-
07	8.6	2^+	7.8	1^-	5.8	2^+	5.1	1^-	2.8	2^+	2.7	1^-
08	9.3	2^+	8.8	1^-	5.5	2^+	5.1	1^-	3.8	2^+	3.7	1^-
09	9.9	2^+	9.3	1^-	5.6	2^+	5.1	1^-	4.3	2^+	4.2	1^-
10	10	2^+	9.9	1^-	6.1	2^+	6	1^-	3.9	2^*	3.9	1^*
11	11.1	2^+	11	1^-	6.2	2^+	6	1^-	5	2^*	5	1^*
12	9.6	2^+	9.5	1^-	6.1	2^+	6	1^-	3.5	2^*	3.5	1^*
13	10.7	2^*	10.6	1^*	6.1	2^+	6	1^-	4.5	2^*	4.6	1^*
14	9.8	2^+	9.5	1^-	6.3	2^+	6	1^-	3.5	2^-	3.6	1^+
15	10.9	2^+	10.8	1^-	6.3	2^+	6	1^-	4.7	2^-	4.8	1^+
16	10.9	2^-	11.2	1^+	6.5	2^-	6.6	1^+	4.4	2^*	4.5	1^*
17	10.8	2^*	10.7	1^*	6.5	2^-	6.7	1^+	4.3	2^+	4	1^-
18	10.8	2^-	11.1	1^+	6.5	2^-	6.7	1^+	4.3	2^-	4.4	1^+

TABLE 6.5. Comparison of the EDO and QD (EA used as the KP operator). The notations are in line with Table 6.1.

Int	EDO (1)		QD (2)		EDO (1)		QD (2)		EDO (1)		QD (2)	
	H	Stat	H	Stat	H_e	Stat	H_e	Stat	H_i	Stat	H_i	Stat
01	9	2^+	8.1	1^-	5.8	2^+	5.1	1^-	3.2	2^+	3	1^-
02	9.7	2^+	9.1	1^-	5.5	2^+	5.1	1^-	4.2	2^+	3.9	1^-
03	10.1	2^+	9.5	1^-	5.5	2^+	5.1	1^-	4.6	2^+	4.4	1^-
04	9	2^+	7.7	1^-	5.8	2^+	5.2	1^-	3.2	2^+	2.5	1^-
05	9.6	2^+	8.7	1^-	5.5	2^+	5.1	1^-	4.1	2^+	3.5	1^-
06	10.2	2^+	9.2	1^-	5.6	2^+	5.1	1^-	4.6	2^+	4.1	1^-
07	8.5	2^+	7.8	1^-	5.8	2^+	5.1	1^-	2.8	2^+	2.7	1^-
08	9.2	2^+	8.8	1^-	5.4	2^+	5.1	1^-	3.8	2^+	3.7	1^-
09	9.9	2^+	9.3	1^-	5.5	2^+	5.1	1^-	4.3	2^+	4.2	1^-
10	10	2^+	9.9	1^-	6	2^+	6	1^-	4	2^+	3.9	1^-
11	11.1	2^*	11	1^*	6	2^*	6	1^*	5	2^*	5	1^*
12	9.6	2^*	9.5	1^*	5.9	2^-	6	1^+	3.7	2^+	3.5	1^-
13	10.6	2^*	10.6	1^*	5.9	2^-	6	1^+	4.7	2^+	4.6	1^-
14	9.7	2^+	9.5	1^-	6.1	2^+	6	1^-	3.5	2^*	3.6	1^*
15	10.9	2^+	10.8	1^-	6.3	2^+	6	1^-	4.6	2^-	4.8	1^+
16	11	2^-	11.2	1^+	6.4	2^-	6.6	1^+	4.6	2^*	4.5	1^*
17	10.9	2^+	10.7	1^-	6.4	2^-	6.7	1^+	4.5	2^+	4	1^-
18	10.8	2^-	11.1	1^+	6.4	2^-	6.7	1^+	4.4	2^-	4.4	1^+

TABLE 6.6. Comparison of the robustness of the populations obtained from the EDO-based EA (1) and the QD-based EA (2). The E and I denotes the percentage of times the population has at least one alternative for the eliminated edges and item, respectively. The Stat notations are in line with Table 6.1.

Int	EDO (1)		QD (2)		EDO (1)		QD (2)	
	E	Stat	E	Stat	I	Stat	I	Stat
1	99.6	2^+	87.5	1^-	70	2^+	51.6	1^-
2	98.2	2^+	92	1^-	60.7	2^+	43.9	1^-
3	97.3	2^+	85.5	1^-	56.2	2^+	43.6	1^-
4	99.8	2^+	90.2	1^-	51.2	2^+	36.8	1^-
5	99.8	2^+	89	1^-	41.6	2^+	32.5	1^-
6	99.6	2^+	86.9	1^-	43.8	2^+	33.2	1^-
7	98.4	2^+	90.2	1^-	30.4	2^+	26.4	1^-
8	98.2	2^+	85.1	1^-	28.7	2^+	22.5	1^-
9	99	2^+	89.8	1^-	28.6	2^+	26.1	1^-
10	64.1	2^+	54.4	1^-	27.7	2^-	31.1	1^+
11	61.5	2^+	49.9	1^-	30.8	2^*	34.9	1^*
12	67	2^+	54.6	1^-	25.4	2^-	28.9	1^+
13	68.8	2^+	54	1^-	23.5	2^-	26.2	1^+
14	67.8	2^+	47	1^-	7.7	2^-	15.5	1^+
15	71.4	2^+	52.8	1^-	7.7	2^-	18.4	1^+
16	29	2^-	67.4	1^+	24.8	2^-	35.8	1^+
17	31	2^-	74.4	1^+	31	2^+	26.5	1^-
18	33.4	2^-	76.3	1^+	16.7	2^-	22	1^+

converge in 10000 iterations for the test instance 16 (a280_n279_bounded-strongly-corr_01). Overall, if we aim to increase the total or edge diversity, using H_e as the fitness function would be better. This is because we achieve similar total diversity with the use of H_e, but it results in higher entropy in the edges, and more importantly, it requires less calculation. However, H works best if we focus on the diversity of items or a more balanced diversity between items and edges.

Now, we conduct the same experiments with $(1+1)$EA to observe the changes in the results. Table 6.3 summarises the results for this round of experiments. Here, one can observe that using H slightly outperforms H_e if we aim for total diversity. The underlying reason is that DP is an exact algorithm that results in the same packing list for identical tours. Thus, identical tours have no contribution to the diversity of edges or items. This is while the $(1+1)$EA can return different packing lists for identical tours and contribute to the diversity of items and overall diversity. Thus, overall diversity is slightly higher when H is used as the fitness function when we incorporate $(1+1)$EA as the inner algorithm.

6.4.3 Comparison of EDO and QD

We compare the introduced EDO-based framework with the QD-based EA in this section. We first run the QD-based EA for 10000 iterations. Then, we set the quality threshold to the minimum quality found in the population obtained by the EA and set μ to the size of the set of solutions obtained. Having set the input parameters, we run the introduced EDO-based algorithm for the same number of iterations. Finally, we compare the two populations in terms of structural diversity (H, H_e, and H_i).

In line with the previous section, we first employ DP as the KP operator; then, $(1+1)$EA is replaced with DP to analyse the impact of using different KP search operators in the results. Table 6.4 shows the results when DP is employed. Compared to the QD-based algorithm, the introduced EA results in a higher H, H_e, and H_i in 14, 15, and 9 cases out of 18, respectively. Table 6.5 summarises the results when $(1+1)$EA is used as the KP operator. The results are almost in line with Table 6.4. Here, the performance of EDO-based EA improves in increasing the entropy of items while it deteriorates in overall and edge diversity. The table shows that the number of cases in favour of EDO-based EA increases to 13 cases taking H_i into account. On the other hand, there is a fall of 1 and 3 cases in terms of H and H_i, respectively.

Furthermore, we conduct an experiment to test the robustness of populations obtained from the EDO and QD-based EAs against changes in the availability of edges and items. In this series of experiments, we make an edge of the best solution of the population unavailable and look into the population to check if there is a solution not using the excluded edge. For items, we look for solutions that behave the opposite of the best solution. For example, if item i is included in the packing list of the best solution, we check if there is a solution excluding item i, and vice versa. We repeat the experiments for all edges and items of the best solution. Table 6.6 summarises the results of the robustness experiment. The results show the EDO-based EA results in more robust sets of solutions. In the edges entropy H_e, it strongly outperforms the QD-based EA in 15 out of 18 test instances, while the figure is 10 for the entropy of items H_i. The results of the small instances (the first 9) where the EDO-based EA converges in 10000 iterations indicate that EDO-based EA can provide a highly robust set of solutions if given sufficient time. Also, we can improve the robustness in edges if we alter the focus on the diversity of edges by using H_e as the fitness function; however, the robustness in items is likely to decrease in this case.

6.5 Co-Evolutionary Algorithm

In Chapter 5, we scrutinise TTP in terms of QD, while Chapter 6 looked into the EDO perspective of the problem. This section presents a co-evolutionary algorithm – outlined in Algorithm 20 – to simultaneously tackle QD and EDO problems in the context of TTP. The algorithm involves two populations P_1 and P_2, employing MAP-Elite-based and EDO-based selection procedures. In other words, P_1 explores niches in the behavioural space, and the P_2 maximises its structural diversity subject to a quality constraint. In QD, a behavioural descriptor (BD) is defined to determine to which part of the behavioural space a solution belongs. In line with Section 5.3, we consider the length of tours $f(x)$, and the profit of selected items $g(y)$, to serve as the BD. To explore niches in the behavioural space, we propose a MAP-Elites-based approach in the next section.

For maximising structural diversity, we utilise the overall entropy measure introduced in section 6.2. Overall, We maximise the solutions' quality and their diversity in the feature space through P_1, while we utilise P_2 to maximise the structural diversity.

6.5.1 Parent Selection and Operators

A bi-level optimisation procedure is employed to generate offspring. A new tour is generated by crossover at the first level; then, (1+1) EA is run to optimise the packing list for the tour. The crossover is the only bridge between P_1 and P_2. For the first parent, we first select P_1 or P_2 uniformly at random. Then, one individual, (x_1, y_1) is selected again uniformly at random from the chosen population; the same procedure is repeated for the selection of the second parent (x_2, y_2). To generate a new solution (x', y') from (x_1, y_1) and (x_2, y_2), a new tour $x' \leftarrow \mathit{crossover}(x_1, x_2)$ is first generated by EAX-1AB crossover same as the previous section. Then, the $(1 + 1)$EA 5.3.2 is initialised to compute a desirable packing list.

6.5.2 Survival Selection Procedures

In this chapter, we discretize P_1 in the behavioural space in the same way in Section5.3. After generating a new solution (x', y'), we find the cell corresponding with its BD

Algorithm 20 The Co-Evolutionary Diversity Algorithm

```
1: Find the optimal/near-optimal values of the TSP and the KP by algorithms in [72, 111], respectively.
2: Generate an empty map and populate it with the initialising procedure.
3: while termination criterion is not met do
4:     Select two individuals based on the parent selection procedure and generate offspring by EAX and $(1 + 1)$ EA.
5:     if The offspring's TSP and the KP scores are within $\alpha_1$, and $\alpha_2$ thresholds to the optimal values of BD. then
6:         Find the corresponding cell to the TSP and the KP scores in the QD map.
7:         if The cell is empty then
8:             Store the offspring in the cell.
9:         else
10:            Compare the offspring and the individual occupying the cell and store the best individual in terms of TTP score in the cell.
11:    if The offspring complies with the quality criterion then
12:        Add the offspring to the EDO population.
13:        if The size of EDO population is equal to $\mu + 1$ then
14:            Remove one individual from the EDO population with the least contribution to diversity.
```

$(f(x'), g(y'))$; if the cell is empty, solution (x', y') is added to the cell. Otherwise, the solution with the highest TTP value is kept in the cell.

Having defined the survival selection of P_1, we now look at P_2's survival selection based on EDO. We add (x', y') to P_2 if the quality criterion is met, i. e., $z(x', y') \geq z_{\min}$. If $|P_2| = \mu + 1$, a solution with the least contribution to $H(P)$ will be discarded.

6.5.3 Self Adaptation

Generating offspring includes the internal (1+1) EA to compute a high-quality packing list for the generated tour. In [88], the $(1 + 1)$ EA is terminated after a fixed number of $t = 2m$ fitness evaluations. However, improving the quality of solutions is easier in the beginning and gets more difficult as the search goes on. Thus, we adopt a similar self-adaptation method proposed in [26, 80] to adjust t during the search. Let $Z = \arg\max_{(x,y)\in P_1}\{z(x, y)\}$. Success defines an increase in Z. We discretize the search to intervals of u fitness evaluations. An interval is successful if Z increases; otherwise, it is a failure. We reset t after each interval; t decreases if Z increases during the last interval. Otherwise, t increases to give the internal $(1 + 1)$ EA more budget in the hope of finding better packing lists and better TTP solutions. Here, we set $t = \gamma m$ where γ can take any value in $[\gamma_{\min}, \gamma_{\max}]$. We set

$$\gamma := \max\{\gamma \cdot F_1, \gamma_{\min}\} \text{ and } \gamma := \min\{\gamma \cdot F_2, \gamma_{\max}\}$$

in case of success and failure, respectively. In our experiments, we use $F_1 = 0.5$, $F_2 = 1.2$, $\gamma_{\min} = 1$, $\gamma_{\max} = 10$, and $u = 2000m$ based on preliminary experiments.

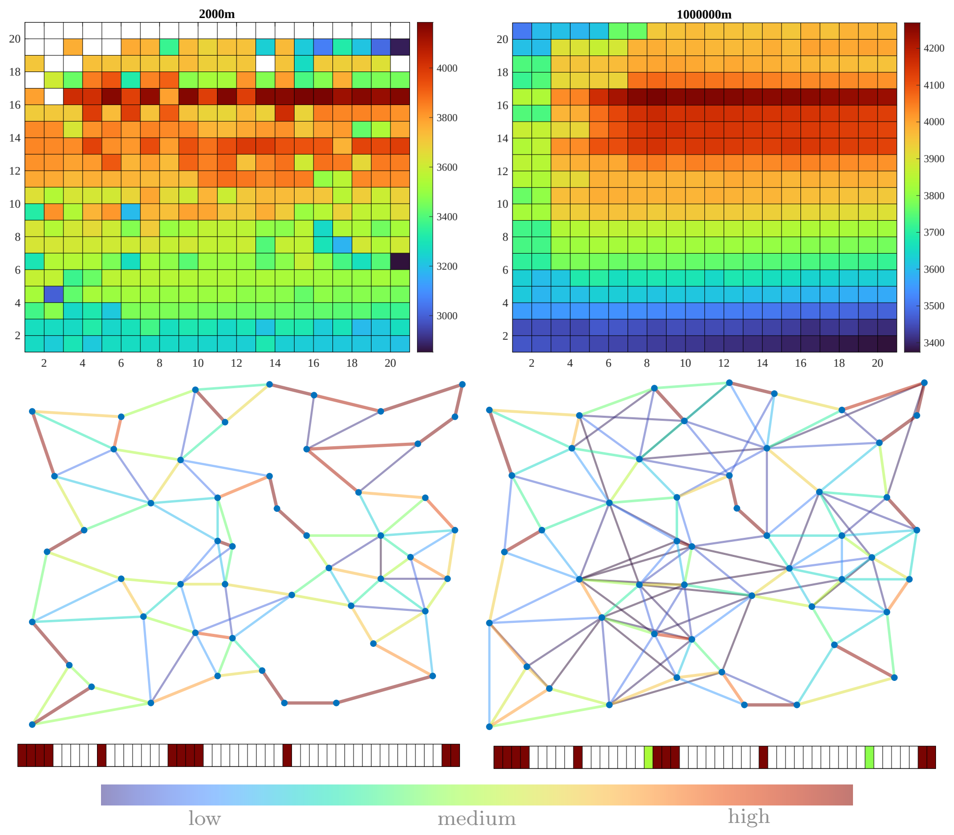


FIGURE 6.2. Evolution of P_1 and P_2 over 4000m and 1000000m fitness evaluations on instance 1 with $\alpha = 2\%$. The first row depicts the distribution of high-quality solutions in the behavioural space (P_1). The second and the third rows show the overlay of all edges and items used in exemplary P_2, respectively. Edges and items are coloured by their frequency.

We refer to this method as $Gamma_1$.

Moreover, we propose an alternative terminating criterion for the internal (1 + 1) EA and denote it $Gamma_2$. Instead of running the (1 + 1) EA for $t = \gamma m$, we terminate (1+1) EA when it fails in improving the packing list in $t' = \gamma' m$ consecutive fitness evaluations. γ' is updated in the same way as γ. Based on the preliminary experiments, we set $\gamma'_{\min}$ and $\gamma'_{\max}$ to 0.1 and 1, respectively.

6.6 Experimental Investigation of Co-EA

We empirically study the Co-EA in this section. We run the Co-EA on eighteen TTP instances from [93]. The same instances are used in the previous chapter. We first illustrate the distribution of solutions in P_1, and the structural diversity of solutions in P_2. Then, we compare the self-adaptation methods with the fixed parameter setting. Afterwards, we conduct a comprehensive comparison between P_1 and P_2 and the populations obtained by the QD-based algorithm and the EDO-based algorithm introduced in the previous chapter and earlier this chapter, respectively. Here, the termination criterion and α are set on 1000000m fitness evaluations and 10%, respectively.

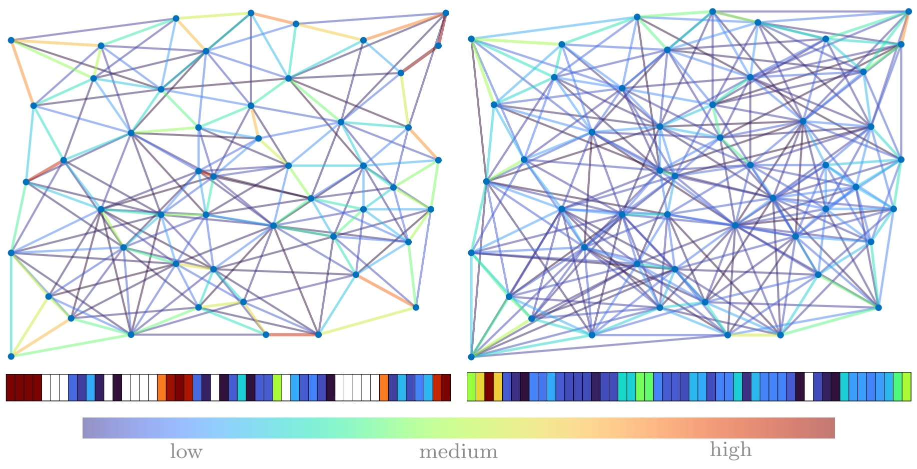


FIGURE 6.3. Overlay of all edges and items used in an exemplary final population P_2 on instance 1 with $\alpha = 10\%$ (left) and $\alpha = 50\%$ (right). Edges and items are coloured by their frequency.

MAP-Elite selection can be beneficial to illustrate the distribution of high-quality solutions in the behaviour space. On the other hand, EDO selection aims to understand which elements in high-quality solutions are easy/difficult to replace. Figure 6.2 depicts exemplary populations P_1 and P_2 after $4000m$ and $1000000m$ fitness evaluations of Co-EA on instance 1, where $\alpha = 2\%$. The first row illustrates the distribution P_1's high-performing solutions over the behavioural space of $f(x)$ and $g(y)$. The second and the third rows represent the overlay of edges and items in P_2, respectively. The figure shows the solutions with the highest quality are located on the top-right of the map on this test instance where the gaps of $f(x)$ and $g(y)$ to f^* and g^* are in [0.0150.035] and [0.150.18], respectively. In the second row of the figure, we can observe that Co-EA successfully incorporates new edges into P_2 and reduces the edges' frequency within the population. However, it is unsuccessful in incorporating new items in P_2. The reason can be that there is a strong correlation between items in this particular test instance, and the difference in the weight and profit of items is significant. It means that there is not many other good items to be replaced with the current selection. Thus, we cannot change the items easily when the quality criterion is fairly tight ($\alpha = 2\%$). As shown on the third row of the figure, the algorithm can change i_8 with i_{43} in some packing lists.

Figure 6.3 reveals that, as α increases, so does the room to involve more items and edges in P_2. In other words, there can be found more edges and items to be included in P_2. Figure 6.3 shows the overlays on the same instances, where α is set to 10% (left) and 50% (right). Not only more edges and items are included in P_2 with the increase of α, but also Co-EA reduces the frequency of the edges and items in P_2 to such a degree that we can barely see any high-frequent edges or items in the figures associated with $\alpha = 50\%$. Moreover, the algorithm can successfully include almost all items in P_2 except item i_{39}. Checking the item's weight, we notice that it is impossible to incorporate the item into any solution. This is because $w_{i_{39}} = 4\,400$,

TABLE 6.7. Comparison of $Gamma_1$ (1) and $Gamma_2$ (2), and the fixed method (3). In columns Stat, the notation X^+ means the median of the measure is better than the one for variant X, X^- means it is worse, and X^* indicates no significant difference. Stat shows the results of the Kruskal-Wallis statistical test at a significance level of 5% and Bonferroni correction. Also, $(x,y)^* = \max_{(x,y)\in P_1}\{z(p)\}$.

	$H(P_2)$						$z((x,y)^*)$					
Inst.	$Gamma_1$	(1)	$Gamma_2$	(2)	$fixed$	(3)	$Gamma_1$	(1)	$Gamma_2$	(2)	$fixed$	(3)
	mean	Stat	mean	Stat	mean	Stat	mean	Stat	mean	Stat	mean	Stat
1	8.7	2^-3^+	8.9	1^+3^+	8.2	1^-2^-	4262.8	2^*3^*	4269.4	1^*3^*	4269.4	1^*2^*
2	9.3	2^*3^+	9.4	1^*3^+	9.1	1^-2^-	7245.8	2^*3^*	7241.9	1^*3^*	7228.3	1^*2^*
3	9.9	2^*3^+	9.9	1^*3^+	9.6	1^-2^-	11724.7	2^*3^*	11712.3	1^*3^*	11673	1^*2^*
4	7.6	2^-3^+	7.7	1^+3^+	7.3	1^-2^-	1460	2^*3^*	1460	1^*3^*	1460	1^*2^*
5	8.9	2^-3^+	9	1^+3^+	8.7	1^-2^-	4265.6	2^*3^*	4276.3	1^*3^+	4253.5	1^*2^-
6	9.3	2^*3^+	9.3	1^*3^+	9.1	1^-2^-	5772.6	2^*3^*	5792.2	1^*3^+	5719.8	1^*2^-
7	8	2^-3^+	8.1	1^+3^+	7.5	1^-2^-	2835.6	2^*3^*	2850.2	1^*3^+	2786.1	1^*2^-
8	8.9	2^-3^+	9	1^+3^+	8.7	1^-2^-	6854.3	2^-3^*	6884.4	1^+3^*	6864.8	1^*2^*
9	9.4	2^-3^+	9.5	1^+3^+	9.2	1^-2^-	11753.2	2^*3^*	11753.2	1^*3^*	11753.2	1^*2^*
10	10.5	2^-3^+	10.6	1^+3^+	10.2	1^-2^-	11135.9	2^*3^*	11137.8	1^*3^*	11137	1^*2^*
11	11.3	2^-3^+	11.4	1^+3^+	11.1	1^-2^-	25553.5	2^*3^*	25508.2	1^*3^*	25505.6	1^*2^*
12	9.3	2^*3^*	9.3	1^*3^*	9.2	1^*2^*	3544.7	2^*3^*	3562.9	1^*3^*	3554.3	1^*2^*
13	10.6	2^*3^+	10.7	1^*3^+	10.5	1^-2^-	13283	2^*3^*	13294.3	1^*3^*	13330.2	1^*2^*
14	9.7	2^*3^+	9.8	1^*3^+	9.5	1^-2^-	5371.7	2^*3^*	5393.6	1^*3^*	5398.3	1^*2^*
15	10.9	2^-3^+	11	1^+3^+	10.7	1^-2^-	20456.8	2^*3^*	20456.8	1^*3^*	20456.8	1^*2^*
16	11.6	2^-3^*	11.7	1^+3^+	11.4	1^*2^-	18388.9	2^-3^*	18474.7	1^+3^*	18399.3	1^*2^*
17	11.2	2^-3^+	11.3	1^+3^+	11.1	1^-2^-	9195.7	2^-3^*	9277.2	1^+3^*	9258.9	1^*2^*
18	11.4	2^-3^+	11.5	1^+3^+	11.1	1^-2^-	19554.4	2^-3^*	19759.5	1^+3^+	19588.5	1^*2^-

while the capacity of the knapsack is set to 4 029. In other words, $w_{i_{39}} > W$.

6.6.1 Analysis of Self-Adaptation

In this sub-section, we compare the two proposed termination criteria and self-adaptation methods $Gamma_1$ and $Gamma_2$ with the fixed method employed in the QD-based algorithm. We incorporate these methods into the Co-EA and run it for ten independent runs. Table 6.7 summarises the mean of P_2's entropy obtained from the competitors. The table indicates that both $Gamma_1$ and $Gamma_2$ outperform the fixed method on all test instances. Kruskal-Wallis statistical tests at significance level 5% and Bonferroni correction also confirm a meaningful difference in the median of results for all instances except instance 12, where there is no significant difference in the mean of $Gamma_1$, $Gamma_2$ and the fixed method. In comparison between $Gamma_1$ and $Gamma_2$, the latter outperforms the first in 15 test instances in terms of the mean of entropy. Moreover, the statistical test i indicates the superiority of $Gamma_2$ in 12 instances. In conclusion, Table 6.7 indicates that $Gamma_2$ works the best with respect to the entropy of P_2.

Moreover, Table 6.7 also shows the mean TTP score of the best solution in P_1 obtained from the three competitors. Table 6.7 indicates that the statistical test confirms a significant difference in the best TTP scores in favour of $Gamma_2$ method in four instances. If we look at the average TTP scores, the results' $Gamma_2$ are slightly better in 9 cases, while $Gamma_1$ and $fixed$ have better results in 3 and 2

Table 6.8. Comparison of the Co-EA and QD from [88] in terms of $z((x,y)^*)$, and EDO algorithm from [87] in $H(P_2)$. Stat shows the results of the Mann-Whitney U-test at significance level 5%. The notations are in line with Table 6.7.

Inst.	Co-EA	(1)	QD	(2)	Co-EA	(1)	EDO	(2)
	Q	Stat	Q	Stat	H	Stat	H	Stat
1	4269.4	2^+	4269.4	1^-	8.9	2^*	8.6	1^*
2	7241.9	2^*	7230.8	1^*	9.4	2^*	9.3	1^*
3	11712.3	2^*	11672.7	1^*	9.9	2^+	9.7	1^-
4	1460	2^+	1460	1^-	7.7	2^+	7.6	1^-
5	4276.3	2^*	4270.5	1^*	9	2^+	8.9	1^-
6	5792.2	2^+	5792.2	1^-	9.3	2^+	9.2	1^-
7	2850.2	2^*	2834.2	1^*	8.1	2^+	8	1^-
8	6884.4	2^*	6876.2	1^*	9	2^+	8.9	1^-
9	11753.2	2^+	11753.2	1^-	9.5	2^+	9.4	1^-
10	11137.8	2^-	11140.2	1^+	10.6	2^+	10.2	1^-
11	25508.2	2^*	25518.4	1^*	11.4	2^+	11.1	1^-
12	3562.9	2^*	3549.9	1^*	9.3	2^+	8.9	1^-
13	13294.3	2^*	13291.7	1^*	10.7	2^+	10.2	1^-
14	5393.6	2^*	5404.4	1^*	9.8	2^+	9.5	1^-
15	20456.8	2^+	20456.8	1^-	11	2^+	10.7	1^-
16	18474.7	2^*	18458.5	1^*	11.7	2^+	11.1	1^-
17	9277.2	2^*	9254.1	1^*	11.3	2^+	10.4	1^-
18	19759.5	2^+	19553.5	1^-	11.5	2^+	11	1^-

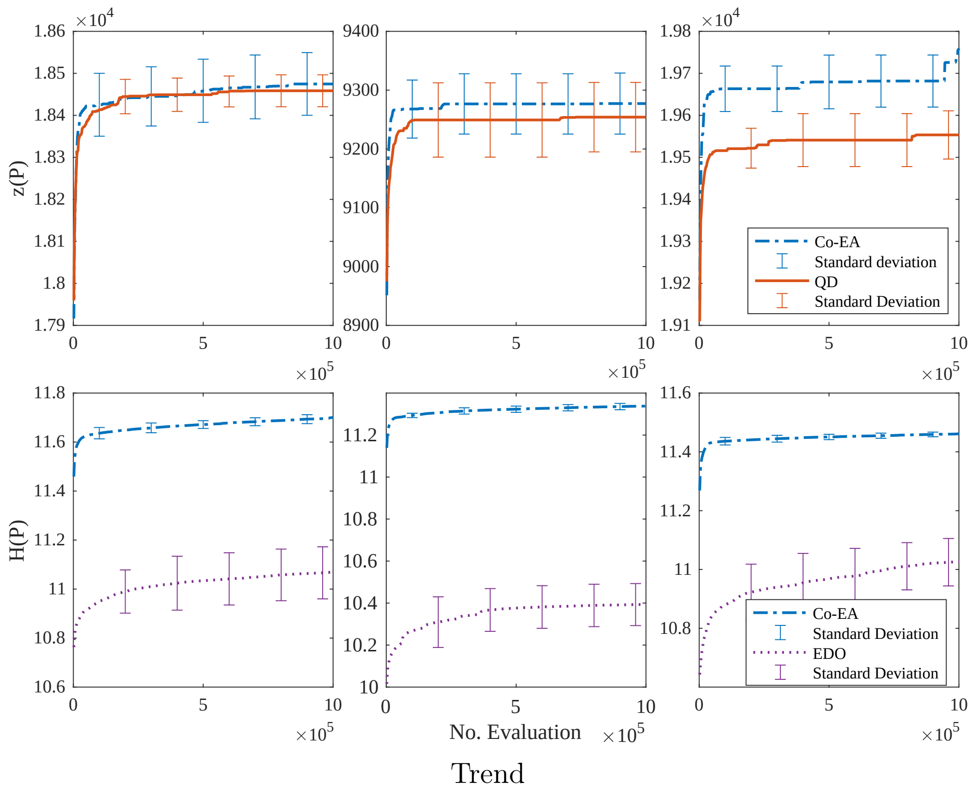


Figure 6.4. Representative trajectories of Co-EA and standard EDO EA on instances 16, 17, 18. The top row shows $H(p_2)$ while the second row shows the best solution in P_1.

cases, respectively. Overall, $Gamma_2$ outperforms the other methods in both entropy and TTP scores. We employ it for the Co-EA in the rest of the study.

6.6.2 Analysis of Co-EA

This section compares P_1 and P_2 with the QD-based EA and the standard EDO algorithm, respectively. Table 6.8 summarises this series of experiments. The results indicate that the Co-EA outperforms the standard EDO in all instances. Except for the first two test instances, the statistical test confirms a significant difference between the EDO algorithm and Co-EA. Moreover, the Co-EA yields competitive results in terms of the quality of the best solution compared to the QD-based EA; in fact, the Co-EA results in a higher mean of TTP scores on 10 test instances. For example, the best solutions found by Co-EA score 19759.5 on average in instance 18, whereby the figure stands at 19553.5 for the QD-based algorithms. The statistical test shows the superiority of CO-EA in four instances over the QD, while the latter algorithm outperforms the prior one in only one instance.

Figure 6.4 depicts the trajectories of Co-EA and the standard EDO algorithm in entropy of the population (the first row) and that of Co-EA and QD-based EA in quality of the best solution (the second row). Note that in the first row, the x-axis shows fitness evaluations from $4000m$ to $1000000m$. This is because P_2 is empty in the early stages of running Co-EA, and we cannot calculate the entropy of P_2 until $|P_2| = \mu$ for the sake of fair comparison. The figure shows that Co-EA converges faster and to a higher entropy than the standard EDO algorithm. Moreover, it also depicts results obtained by Co-EA have much less standard deviation. Regarding the quality of the best solution, both Co-EA and QD-based EA follow a similar trend, but the prior algorithm converges to a higher TTP value.

6.7 Conclusion

We introduced a framework to generate a set of high-quality TTP solutions differing in structural diversity. We examined the inter-dependency of TTP's sub-problems, TSP and KP, and determined the best method to achieve a highly diverse set of solutions. Moreover, We empirically analysed the introduced framework and compared the results with a recently developed QD-based algorithm in terms of diversity. The results showed a considerable improvement in the diversity of the population compared to the QD-based algorithm. We conduct a simulation test to evaluate the robustness of the population obtained from the two frameworks. We also introduced a co-evolutionary algorithm to simultaneously evolve two populations for the travelling thief problem. The first population explore niches in a behavioural space, and the other maximises structural diversity. The results showed the superiority of the algorithm to the standard framework in the literature in maximising diversity. The co-evolutionary algorithm also yields competitive results in terms of quality.

For future study, it is intriguing to incorporate indicators from multi-objective optimisation frameworks into the algorithm to focus on diversities of edges and items and compare them to the incumbent method. It is also interesting to adopt a more complicated MAP-Elites-based survival selection for exploring the behavioural space. Also, other co-evolutionary algorithms in the literature can be studied to make a comparison to the introduced algorithms. Moreover, this study can be a transition from benchmark problems to real-world optimisation problems where imperfect modelling is common, and diversity in solutions can be beneficial.

Chapter 7

Analysis of Quality Diversity in the Knapsack Problem

7.1 Introduction

In Chapters 5 and 6, we conducted comprehensive empirical investigations into the application of QD in the context of TTP. These empirical results demonstrate the efficiency of QD in the diversification of TTP solutions and solving the TTP. Having studied QD empirically, we now aim to provide a theoretical analysis of QD to expand our understating of these methods in combinatorial problems. We seek to comprehensively understand how these approaches' search mechanism works and how it leads to high-quality solutions.

In this chapter, we scrutinise QD-based algorithms regarding runtime for the first in the literature. The runtime analysis is the computational examination of algorithms taking into account the number of evaluations required to achieve the optimal solution with respect to the size of the input. The field began with [67] studying a basic EA in a number of simple binary functions such as ONEMAX. Several advancements and theoretical tools such as fitness-based partitions [104] and Drift analysis [47] have been developed that can become handy in performing runtime analysis of bio-inspired algorithms. In recent years, several studies conducted runtime analysis and drew a comparison among EAs in various problems [27, 37, 122]. We refer readers interested in recent developments of runtime analysis to [30].

We contribute to the research line of evolutionary diversity by theoretically and empirically studying QD for the KP, with a focus on connections between populating behavioural spaces and constructing solutions in a dynamic programming (DP) manner. The use of evolutionary algorithms building populations of specific structures to carry out dynamic programming has been studied in [28, 50, 110]. We consider mimicking dynamic programming behaviour by using QD algorithms with appropriately defined behavioural spaces. To this end, we define two behavioural spaces based on weights, profits and the subset of the first i items, as inspired by dynamic programming (DP) [111] and the classic fully polynomial-time approximation scheme (FPTAS) [114]. Here, the scaling factor used in the FPTAS adjusts the niche size along the weight/profit dimension. We formulate two simple mutation-only algorithms based on MAP-Elite to populate these spaces. We show that both algorithms mimic DP and find an optimum within the pseudo-polynomial expected runtime. Moreover, we show that in the profit-based space, the algorithm can be made into a fully polynomial-time randomised approximation scheme (FPRAS) with an appropriate choice of scaling value. Our experimental investigation on various instances suggests that these algorithms significantly outperform $(1+1)$EA and $(\mu+1)$EA, especially in hard cases. With this, we demonstrate the ability of QD-based mechanisms to imitate DP-like behaviours in KP and thus its potential value in black-box optimisers for problems with recursive subproblem structures.

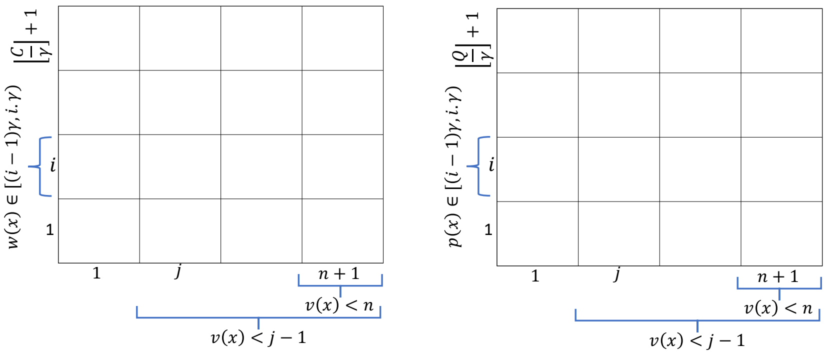
The work of this chapter is based on a conference paper [84] presented at the parallel problem solving from nature (PPSN 2022). The remainder of this chapter is structured as follows. We formally define the knapsack problem, the behavioural spaces, and the algorithms in 7.2. Next, we provide a runtime analysis for the algorithms in 7.3. In Section 7.4, we examine the distribution of high-quality knapsack solutions in the behavioural spaces and compare QD-based algorithms to other EAs. Finally, we finish with some concluding remarks.

7.2 Quality-Diversity for the knapsack problem

As described in Section 2.4.1, the knapsack problem is defined on a set of items I, where $|I| = n$ and each item i corresponds to a weight w_i and a profit p_i. Here, the goal is to find a selection of item $x = (x_1, x_2, \dots, x_n)$ that maximises the profit while the weight of selected items is constrained to a capacity W. Here, x is the characteristic vector of the selection of items. Technically, KP is a binary linear programming problem: let $w = (w_1, \dots, w_n)$ and $p = (p_1, \dots, p_n)$, find $\arg\max_{x \in \{0,1\}^n} \left\{ p^T x \mid w^T x \leq W \right\}$. We assume that all items have weights in $\{0, 1, \cdots, W]$, since any item violating this can be removed from the problem instance.

In this section, we introduce two MAP-Elite-based algorithms exploring two different behavioural spaces. To determine the behaviour of a solution in a particular space, a behaviour descriptor (BD) is required. MAP-Elite is an EA, where a solution competes with other solutions with a similar BD. MAP-Elites algorithm discretises a behavioural space into a grid to define the similarity and acceptable tolerance of difference in two descriptors. Each cell in the grid corresponds with a BD type, and only the best solution with that particular BD is kept in the cell.

For KP, we formulate the behavioural spaces based on the two ways in which the classic dynamic programming approach is implemented [111], i.e. profit-based and


$\lfloor \frac{C}{\gamma} \rfloor + 1$
$w(x) \in [(i-1)\gamma, i.\gamma)$
i
1
1
j
$n+1$
$v(x) < n$
$v(x) < j-1$
$\lfloor \frac{Q}{\gamma} \rfloor + 1$
$p(x) \in [(i-1)\gamma, i.\gamma)$
i
1
1
j
$n+1$
$v(x) < n$
$v(x) < j-1$

(A) Weight-based (B) Profit-based

Figure 7.1. The representation of the empty maps in the behavioural spaces.

weight-based sub-problem divisions. Let $v(x)$ be the function returning the index of the last item in solution x: $v(x) = \max_i\{i \mid x_i = 1\}$.

7.2.1 Weight-based space

For the weight-based approach, $w(x)$ and $v(x)$ serve as the BD, where $w(x) = w^T x$. Figure 7.1a outlines an empty map in the weight-based behavioural space. To exclude infeasible solutions, the weight dimension is restricted to $\{0, 1, \cdots W\}$. As depicted, the behavioural space consists of $(\lfloor W/\gamma \rfloor + 1) \times (n+1)$ cells, in which cell (i, j) includes the best solution x (i.e. maximizing $p(x) = p^T x$) with $v(x) = j - 1$ and $w(x) \in [(i-1)\gamma, i\gamma)$. Here, γ is a factor to determine the size of each cell. The algorithm is initiated with a zero string 0^n. Having a parent is selected uniformly at random from the population, we generate a single offspring by standard flip mutation. If $w(x) \leq W$, we find the cell corresponding with the solution BD. We check the cells one by one. If the cells are empty, x is stored in the cell; otherwise, the solution with the highest profit remains in the corresponding cell. These steps are continued until a termination criterion is met.

7.2.2 Profit-based space

For the profit-based approach, $p(x)$ and $v(x)$ serve as the BD. Figure 7.1b depicts the profit-based behavioural space with $(\lfloor Q/\gamma \rfloor + 1) \times (n+1)$ cells where $Q = \sum_{i \in I} p_i$. Here, the selection in each cell minimizes the weight, and cell (i, j) includes a solution x with $v(x) = j - 1$ and $p(x) \in [(i-1)\gamma, i\gamma)$. Otherwise, the parent selection and the operator are the same as in weight-based MAP-Elite. After generating the offspring, we determine the cell associating with the BD $((v(x), p(x)))$. If the cell is empty, the solution is stored in the cells; otherwise, the solution with the lower weight $w(x)$ is kept in the cell. The steps are continued until a termination criterion is met.

7.2.3 DP-based filtering scheme

In classical MAP-Elites, the competition between solutions is confined within each cell. However, in this context, the mapping from solution space to behaviour space

Algorithm 21 weight-based MAP-Elites

```
Require: weights $\{w_i\}_{i=1}^n$, $W$, profits $\{p_i\}_{i=1}^n$, $\gamma$
 1: $P \leftarrow \{0^n\}$                         // $P$ is indexed from 1, $0^n$ is an all-zero string
 2: $A \leftarrow 0_{n+1\times\lfloor W/\gamma\rfloor+1}$          // $0_{n+1\times\lfloor W/\gamma\rfloor+1}$ is an all-zero matrix
 3: $B \leftarrow 0$,
 4: while Termination criteria are not met do
 5:   $i \leftarrow Uniform(\{1,\dots,|P|\})$
 6:   Get $x$ from flipping each bit in $P(i)$ independently with probability $1/n$
 7:   if $w(x) \le W$ then
 8:     $W' \leftarrow \lfloor w(x)/\gamma\rfloor + 1$
 9:     if $A_{v(x)+1,W'} = 0$ then
10:       $P \leftarrow P \cup \{x\}$                        // $x$ is indexed last in $P$
11:       $A_{v(x)+1,W'} \leftarrow |P|$
12:     else if $p(x) > p(P(A_{v(x)+1,W'}))$ then
13:       $P(A_{v(x)+1,W'}) \leftarrow x$
14:     for $j$ from $v(x)+2$ to $n+1$ do {DP-based filtering scheme}
15:       if $A_{j,W'} = 0$ Or $p(x) > p(P(A_{j,W'}))$ then
16:         $A_{j,W'} \leftarrow A_{v(x)+1,W'}$
17:     if $p(x) > B$ then
18:       $B \leftarrow p(x)$
19: return B
```

is transparent enough in both cases that a dominant relation between solutions in different cells can be determined; a property exploited by the DP approach. Therefore, in order to reduce the population size and speed up the search for the optimum, we incorporate a filtering scheme that forms the core of the DP approach. Given solutions x_1 and x_2 with $v(x_1) \ge v(x_2)$ and $w(x_1) = w(x_2)$; then, x_1 dominates x_2 in the weight-based space if $p(x_1) > p(x_2)$. To filter out the dominated solutions, we relax the restriction that each BD corresponds to only one cell and redefine acceptable solutions for Cell (i, j) in the weight-based space: $v(x) \le j-1$ and $w(x) \in [(i-1)\gamma, i\gamma)$. This means a particular BD is acceptable for multiple cells, and MAP-Elite algorithms must check all the cells accepting the offspring. Algorithm 21 outlines the MAP-Elite algorithm exploring this space; this is referred to as weight-based MAP-Elites.

The same scheme can be applied to the profit-based space, where cell (i, j) accepts solution x with $v(x) \le j - 1$ and $p(x) \in [(i-1)\gamma, i\gamma)$. In this case, the dominance relation is formulated to minimise weight. Algorithm 22 sketches the profit-based MAP-Elites.

7.3 Theoretical analysis

In this section, we give some runtime results for Algorithm 21 and 22 based on expected time, as typically done in runtime analyses. Here, we use "time" as a shorthand for "number of fitness evaluations", which in this case equals the number of generated solutions during a run of the algorithm. We define $a \wedge b$ and $a \vee b$ to be the bit-wise AND and bit-wise OR, respectively, between two equal-length bit-strings a and b.

Algorithm 22 profit-based MAP-Elites

```
Require: Weights $\{w_i\}_{i=1}^n$, $W$, profits $\{p_i\}_{i=1}^n$, $\gamma$
1: $P \leftarrow \{0^n\}$    // $P$ is indexed from 1, $0^n$ is an all-zero string
2: $A \leftarrow 0_{n+1\times\sum_{i=1}^n p_i+1}$    // $0_{n+1\times\lfloor C/\gamma\rfloor+1}$ is an all-zero matrix
3: $B \leftarrow 0$,
4: while Termination criteria are not met do
5:   $i \leftarrow Uniform(\{1,\dots,|P|\})$
6:   Get $x$ from flipping each bit in $P(i)$ independently with probability $1/n$
7:   $G \leftarrow \lfloor p(x)/\gamma\rfloor + 1$
8:   if $A_{v(x)+1,G} = 0$ then
9:     $P \leftarrow P \cup \{x\}$    // $x$ is indexed last in $P$
10:    $A_{v(x)+1,G} \leftarrow |P|$
11:  else if $w(x) < w(P(A_{v(x)+1,G}))$ then
12:    $P(A_{v(x)+1,G}) \leftarrow x$
13:  for $j$ from $v(x)+2$ to $n+1$ do {DP-based filtering scheme}
14:    if $A_{j,G} = 0$ Or $w(x) < w(P(A_{j,G}))$ then
15:      $A_{j,G} \leftarrow A_{v(x)+1,G}$
16:  if $w(x) \le C$ then
17:    if $p(x) > B$ then
18:      $B \leftarrow p(x)$
19: return B
```

Also, we denote k-length all-zero and all-one bit-strings by 0^k and 1^k, respectively. For convenience, we denote the k-size prefix of $a \in \{0,1\}^n$ with $a^{(k)} = a \wedge 1^k0^{n-k}$, and the k-size suffix with $a_{(k)} = a \wedge 0^{n-k}1^k$.

It is important to note that in all our proofs, we consider solution y replacing solution x during a run to imply $v(y) \le v(x)$. Since this holds regardless of whether the filtering scheme outlined in Section 7.2.3 is used, our results should apply to both cases, as we use the largest possible upper bound of population size. Note that this filtering scheme may not reduce the population size in some cases.

We first show that with $\gamma = 1$ (no scaling), Algorithm 21 ensures that prefixes of optimal solutions remain in the population throughout the run and that these increase in sizes within a pseudo-polynomial expected time. For this result, we assume all weights are integers.

Theorem 2. *Given $\gamma = 1$ and $k \in [0,n]$, within expected time at most $e(W+1)n^2k$, Algorithm 21 achieves a population P such that for any $j \in [0,k]$, there is an optimal solution x^* where $x^{*(j)} \in P$.*

Proof. Let P_t be the population at iteration $t \ge 0$, S be the set of optimal solutions, $S_j = \{s^{(j)} \mid s \in S\}$, $X_t = \max\{h \mid \forall j \in [0,h], S_j \cap P_t \ne \emptyset\}$, and $H(x,y)$ be the Hamming distance between x and y, we have $S_n = S$. We see that for any $j \in [0, X_t]$, any $x \in S_j \cap P_t$ must be in $P_{>t}$, since otherwise, let y be the solution replacing it, and $y^* = y \vee x^*_{(n-j)}$ for any $x^* \in S$ where $x = x^{*(j)}$, we would have $p(y^*) - p(x^*) =$

$p(y) - p(x) > 0$ and $w(y^*) = w(x^*) \leq B$, a contradiction. Additionally, if $x \in S_i \cap S_j$ for any $0 \leq i < j \leq n$, then $x \in \bigcap_{h=i}^{j} S_h$. Thus, if $X_t < n$, then $S_{X_t} \cap S \cap P_t = \emptyset$, so for all $x \in S_{X_t} \cap P_t$, there is $y \in S_{>X_t}$ such that $H(x, y) = 1$. We can then imply from the algorithm's behaviour that for any $j \in [0, n-1]$, $Pr[X_{t+1} < j \mid X_t = j] = 0$ and

$$Pr[X_{t+1} > j \mid X_t = j] \geq \frac{1}{n}\left(1 - \frac{1}{n}\right)^{n-1} \frac{|S_{X_t} \cap P_t|}{|P_t|} \geq \frac{1}{en \max_h |P_h|}.$$

Let T be the minimum integer such that $X_{t+T} > X_t$, then the expected waiting time in a binomial process gives $E[T \mid X_t < j] \leq en \max_h |P_h|$ for any $j \in [1, n]$. Let T_k be the minimum integer such that $X_{T_k} \geq k$, we have for any $k \in [0, n]$, $E[T_k] \leq \sum_{i=1}^{k} E[T \mid X_t < i] \leq en \max_h |P_h| k$, given that $0^n \in S_0 \cap P_0$. Applying the bound $\max_h |P_h| \leq (W + 1)n$ yields the claim. □ □

We remark that with $\gamma > 1$, Algorithm 21 may fail to maintain prefixes of optimal solutions during a run due to rounding error. That is, assuming there is $x = x^{*(j)} \in P_t$ at step t and for some $j \in [0, n]$ and optimal solution x^*, a solution y may replace x if $p(y) > p(x)$ and $w(y) < w(x) + \gamma$. It is possible that $y^* = y \vee x^*_{(n-j)}$ is infeasible (i.e. when $W < w(x^*) + \gamma$), in which case the algorithm may need to "fix" y with multiple bit-flips in one step. The expected runtime till optimality can be derived directly from Theorem 2 by setting $k = n$.

Corollary 1. *Algorithm 21, run with $\gamma = 1$, finds an optimum within expected time at most $e(W + 1)n^3$.*

Using the notation $Q = \sum_{i=1}^{n} p_i$, we have the following result for Algorithm 22, which is analogous to Theorem 2 for Algorithm 21.

Theorem 3. *Given $k \in [0, n]$, and let z be an optimal solution, within expected time at most $e\left(\lfloor Q/\gamma \rfloor + 1\right) n^2 k$, Algorithm 22 achieves a population P such that, if $\gamma > 0$ is such that p_i/γ is integer for every item i in z, then for any $j \in [0, k]$, there is a feasible solution x where*

- *there is an integer m such that $p(x^{(j)}), p(z^{(j)}) \in [m\gamma, (m + 1)\gamma)$,*
- *$x_{(n-j)} = z_{(n-j)}$,*
- *$x^{(j)} \in P$.*

Moreover, for other γ values, the first property becomes $p(x^{(j)}), p(z^{(j)}) \in [m\gamma, (m + j + 1)\gamma)$.

Proof. The proof proceeds similarly as that of Theorem 2. We have the claim holds for $k = 0$ since the empty set satisfies the properties for $j = 0$ (i.e. x and z would be the same). For other k values, it suffices to show that if there is such a solution x for some $j \in [0, k]$: 1) any solution y replacing $x^{(j)}$ in a future step must be the j-size prefix of another solution with the same properties, and 2) at most one bit-flip is necessary to have it also holds for $j + 1$.

1) Let y be the solution replacing $x^{(j)}$, we have $p(y), p(x^{(j)}) \in [m\gamma, (m + 1)\gamma)$ for some integer m, and $w(y) < w(x^{(j)})$. Let $y^* = y \vee z_{(n-j)}$, we have $p(y), p(z^{(j)}) \in [m\gamma, (m + 1)\gamma)$, and $w(y^*) - w(x) = w(y) - w(x^{(j)}) < 0$, implying y^* is feasible. Therefore, y^* possesses the same properties as x. Note that this also holds for the case where $p(x^{(j)}), p(z^{(j)}) \in [m\gamma, (m+j+1)\gamma)$. In this case, $p(y) \in [m\gamma, (m+j+1)\gamma)$.

2) If this also holds for $j + 1$, no further step is necessary. Assuming otherwise, then z contains item $j + 1$, the algorithm only needs to flip the position $j + 1$ in $x^{(j)}$, since x and z shares $(n - j - 1)$-size suffix, and the p_{j+1} is a multiple of γ. Since this occurs with probability at least $1/en \max_h |P_h|$, the rest follows identically, save for $\max_h |P_h| \leq (\lfloor Q/\gamma \rfloor + 1)\, n$. If p_{j+1} is a not multiple of γ, then $p(x^{(j+1)})$ may be mapped to a different profit range from $p(z^{(j+1)})$. The difference is increased by at most 1 since $p(x^{(j+1)}) - p(x^{(j)}) = p(z^{(j+1)}) - p(z^{(j)})$, i.e. if $p(x^{(j)}), p(z^{(j)}) \in [m\gamma, (m+j+1)\gamma)$ for some integer m, then $p(x^{(j+1)}), p(z^{(j+1)}) \in [m'\gamma, (m'+j+2)\gamma)$ for some integer $m' \geq m$. Since x can be replaced in a future step by another solution with a smaller profit due to rounding error, the difference can still increase, so the claim holds non-trivially.

□ □

Theorem 3 gives us the following profit guarantees of Algorithm 22 when $k = n$. Here OPT denotes the optimal profit.

Corollary 2. *Algorithm 22, run with $\gamma > 0$, within expected time at most $e\,(\lfloor Q/\gamma \rfloor + 1)\, n^3$ obtains a feasible solution x where $p(x) = OPT$ if p_i/γ is integer for all $i = 1, \dots, n$, and $p(x) > OPT - \gamma n$ otherwise.*

Proof. If p_i/γ is integer for all $i = 1, \dots, n$, then $|p(a) - p(b)|$ is a multiple of γ for any solutions a and b. Since by Theorem 2, x is feasible and $p(x) > OPT - \gamma$, it must be that $p(x) = OPT$. For the other case, Theorem 2 implies that $p(x), OPT \in [m\gamma, (m + n + 1)\gamma)$ for some integer m. This means $p(x) > OPT - \gamma n$. □ □

Using this property, we can set up a FPRAS with an appropriate choice of γ, which is reminiscent of the classic FPTAS for KP based on DP. As a reminder, x is a $(1 - \epsilon)$-approximation for some $\epsilon \in (0, 1)$ if $p(x) \geq (1 - \epsilon)OPT$. The following corollary is obtained from the fact that $Q \leq n \max_i\{p_i\}$, and $\max_i\{p_i\} \leq OPT$.

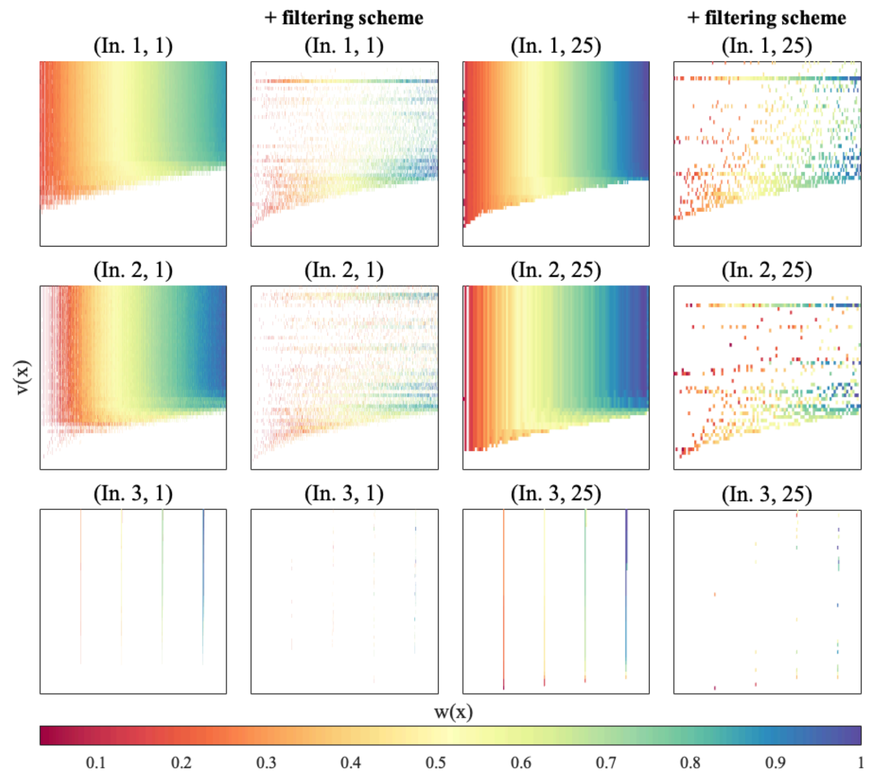


FIGURE 7.2. The distribution of high-performing solutions in the weight-based behavioural space. The title of sub-figures shows (Ints. No, γ). Colors are scaled to OPT.

Corollary 3. *For some $\epsilon \in (0, 1)$, Algorithm 22, run with $\gamma = \epsilon \max_i\{p_i\}/n$, obtains a $(1-\epsilon)$-approximation within expected time at most $e\left(\lfloor n^2/\epsilon \rfloor + 1\right) n^3$.*

For comparison, the asymptotic runtime of the classic FPTAS achieving the same approximation guarantee is $O(n^2\lfloor n/\epsilon \rfloor)$ [114].

7.4 Experimental investigations

In this section, we experimentally examine the two MAP-Elite-based algorithms. The experiments can be categorised into three sections. First, we illustrate the distribution of high-performing solutions in the two behavioural spaces. Second, we compare Algorithm 21 and 22 in terms of population size and ratio in achieving the optimums over 30 independent runs. Finally, we compare between the best MAP-Elite algorithm and two baseline EAs, namely $(1+1)$EA and $(\mu+1)$EA. These baselines are selected due the the same size of offspring in each iteration. For the first round of experiments, three instances from [93] are considered. There is a strong correlation between the weight and profit of each item in the first instance. The second and third instances are not correlated, while the items have similar weights in the third instance. The termination criterion is set to the maximum fitness evaluations of Wn^2. We also set $\gamma \in \{1, 5, 25\}$. For the second and third rounds of experiments, we run algorithms on 18 test instances from [93], and change the termination criterion to either achieve the optimal value or the maximum CPU-time of 7200 seconds.

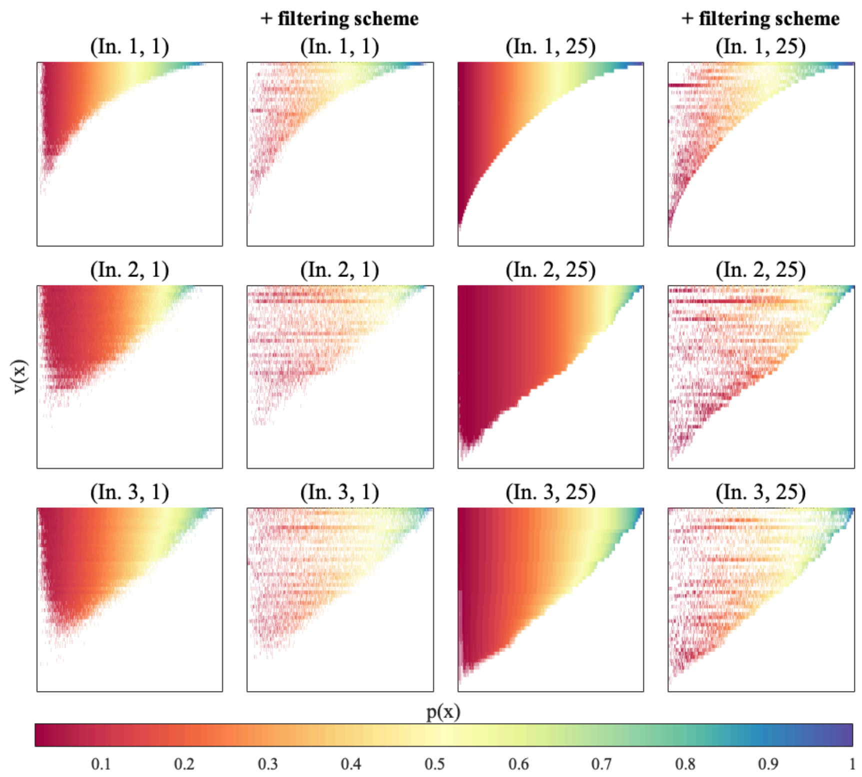


FIGURE 7.3. The distribution of high-performing solutions in the profit-based behavioural space. Analogous to Figure 7.2.

Figure 7.2 illustrates the high-performing solutions obtained by Algorithm 21 in the weight-based space. As shown in the figure, the best solutions can be found at the right top of the space. One can expect it since in that area of the space, solutions get to involve most items and most of the knapsack's capacity, whereby on the left bottom of the space, a few items and a small proportion of C can be used. Algorithm 21 can successfully populate most of the space in instance 1, 2, while we can see most of the space is empty in instance 3. This is because the weights are uniformly distributed within $[1000, 1010]$, while C is set to 4567. As shown in the figure, the feasible solutions can only pick 4 items. Figure 7.2 also shows that the DP-based filtering removes many dominated solutions that contribute to the convergence rate and pace of the algorithm.

Figure 7.3 shows the best-performing solutions obtained by Algorithm 22 in the profit-based space. It can be observed that we can only populate half of the space by Algorithm 22 or any other algorithm. To have a solution with a profit of Q, the solution should include all items. This means that it is impossible to populate any other cells except cell $(n+1), (Q+1)$. On the contrary of the weight-based space, we can have both feasible and infeasible solutions in the profit-based space. For example, the map is well populated in instance 3, but mostly contains infeasible solutions. Figure 7.4 depicts the trajectories of population size of Algorithm 21 and 22. The figure shows Algorithm 21 results in significantly smaller $|P|$ than Algorithm 22. For example, the final population size of Algorithm 21 is equal to 37 in instance 3, where $\gamma = 25$, while it is around 9000 for Algorithm 22. This is because we can limit the first space to

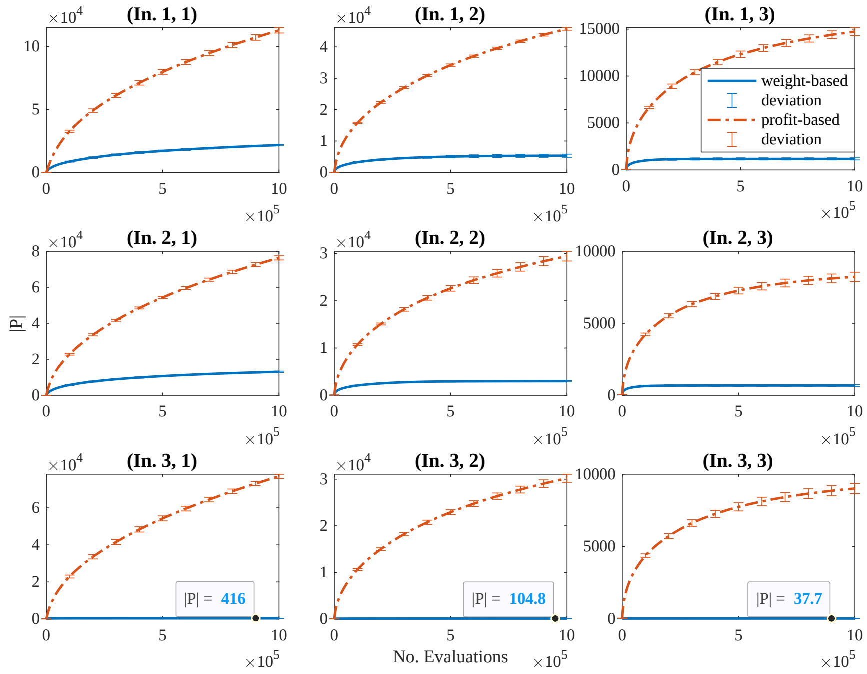


FIGURE 7.4. Means and standard deviations of population sizes over fitness evaluations (the filtering scheme is used).

the promising part of it ($w(x) \leq W$), but we do not have a similar advantage for the profit-based space; the space accepts the full range of possible profits ($p(x) \leq Q$). We believe this issue can cause an adverse effect on the efficiency of MAP-Elites in reaching optimality based on theoretical observations. This is explored further in our second experiment, where we look at the actual run-time to achieve the optimum.

Table 7.1 and 7.2 show the ratio of Algorithm 21 and Algorithm 22 in achieving the optimum for each instance in 30 independent runs, respectively. The tables also present the mean of fitness evaluations for the algorithms to hit the optimal value or reach the limitation of CPU time. Table 7.1 shows that the ratio is 100% for Algorithm 21 on all instances and all $\gamma \in \{1, 5, 25\}$. On the other hand, Algorithm 22 cannot achieve the optimums in all 30 runs, especially in large instances when $\gamma = 1$. However, increasing γ to 25 enables the algorithm to obtain the optimum in most instances, with the exception of instance 9. Moreover, the number of fitness evaluations required for Algorithm 22 is considerably higher than that of Algorithm 21. We can conclude that Algorithm 21 is more time-efficient than Algorithm 22, confirming our theoretical findings. This also suggests that the rounding errors are not detrimental to these algorithms' performances.

For the last round of the experiments, we compare Algorithm 21 to two well-known EAs in the literature, $(1 + 1)$EA and $(\mu + 1)$EA. Table 7.3 presents the ratio of the three algorithms in achieving the optimum and the mean of fitness evaluations required for them to reach the optimum. As shown in the table, the performances of $(1+1)$EA and $(\mu+1)$EA deteriorate on the strongly correlated instances. It seems that $(1+1)$EA and $(\mu+1)$EA are prone to get stuck in local optima, especially in instances with a strong weights-profits correlation. On the other hand, the MAP-Elite algorithm

Table 7.1. Number of fitness evaluations needed by Algorithm 21 to obtain the optimal solutions.

In.	n	C	U	$\gamma = 1$		$\gamma = 5$		$\gamma = 25$	
				mean	time	mean	time	mean	time
1	50	4029	1.37e+09	1.53e+06	2.74e+01	3.76e+05	4.35e+00	1.61e+05	1.96e+00
2	50	2226	7.57e+08	5.32e+05	8.14e+00	1.74e+05	2.32e+00	5.86e+04	9.75e-01
3	50	4567	1.55e+09	2.43e+04	3.48e-01	1.12e+04	1.81e-01	5.82e+03	1.08e-01
4	75	5780	6.63e+09	5.30e+06	8.28e+01	1.45e+06	2.07e+01	4.12e+05	5.60e+00
5	75	3520	4.04e+09	3.63e+06	7.11e+01	1.15e+06	2.44e+01	3.49e+05	5.96e+00
6	75	6850	7.86e+09	1.17e+05	2.21e+00	4.09e+04	5.91e-01	1.44e+04	2.32e-01
7	100	8375	2.28e+10	2.42e+07	4.75e+02	6.57e+06	1.33e+02	6.60e+07	1.34e+03
8	100	4815	1.31e+10	9.56e+06	2.02e+02	2.73e+06	5.07e+01	8.66e+05	1.30e+01
9	100	9133	2.48e+10	6.18e+05	1.06e+01	1.78e+05	3.26e+00	5.79e+04	1.34e+00
10	123	10074	5.10e+10	3.56e+07	6.65e+02	9.90e+06	1.77e+02	2.55e+07	4.71e+02
11	123	5737	2.90e+10	2.05e+07	5.40e+02	5.12e+06	9.38e+01	1.47e+06	3.54e+01
12	123	11235	5.68e+10	1.45e+06	3.74e+01	3.38e+05	5.27e+00	1.21e+05	1.90e+00
13	151	12422	1.16e+11	5.04e+07	9.15e+02	1.48e+07	2.73e+02	7.51e+06	1.86e+02
14	151	6924	6.48e+10	4.27e+07	9.75e+02	1.24e+07	2.73e+02	3.35e+06	5.71e+01
15	151	13790	1.29e+11	3.18e+06	9.87e+01	6.73e+05	1.70e+01	2.35e+05	3.94e+00

Table 7.2. Number of fitness evaluations needed by Algorithm 22 to obtain the optimal solutions

In.	n	Q	$\gamma = 1$			$\gamma = 5$			$\gamma = 25$		
			mean	ratio	U	mean	ratio	U	mean	ratio	U
1	50	53928	1.15e+07	100	1.83e+10	3.68e+06	100	e.66e+09	1.21e+06	100	7.33e+08
2	50	23825	5.36e+06	100	8.10e+09	1.34e+06	100	1.62e+09	4.00e+05	100	3.24e+08
3	50	24491	3.86e+06	100	8.32e+09	1.27e+06	100	1.66e+09	1.27e+06	100	3.33e+08
4	75	78483	5.07e+07	100	9.00e+10	1.50e+07	100	1.8e+10	4.03e+06	100	3.6e+09
5	75	37237	2.74e+07	100	4.27e+10	7.36e+06	100	8.54e+09	2.32e+06	100	1.71e+09
6	75	38724	1.93e+07	100	4.44e+10	5.63e+06	100	8.88e+09	2.95e+06	100	1.78e+09
7	100	112635	2.24e+08	97	3.06e+11	6.67e+07	100	6.12e+10	1.72e+07	100	1.22e+10
8	100	48042	6.76e+07	100	1.31e+11	1.82e+07	100	2.61e+10	5.03e+06	100	5.22e+09
9	100	52967	7.99e+07	100	1.44e+11	2.86e+07	100	2.88e+10	1.18e+08	87	5.76e+09
10	123	135522	3.35e+08	87	6.86e+11	1.05e+08	100	1.37e+11	7.34e+07	100	2.74e+10
11	123	57554	1.47e+08	100	2.91e+11	3.58e+07	100	5.82e+10	8.87e+06	100	1.16e+10
12	123	63116	1.71e+08	97	3.19e+11	8.45e+07	100	5.38e+10	2.66e+07	100	1.28e+10
13	151	166842	3.81e+08	13	1.56e+12	1.39e+08	100	3.12e+11	5.69e+07	100	6.25e+10
14	151	70276	3.15e+08	77	6.58e+11	8.86e+07	100	1.32e+11	1.96e+07	100	2.63e+10
15	151	76171	2.64e+08	90	7.13e+11	9.58e+07	100	1.42e+11	3.16e+07	100	2.85e+10
16	194	227046	2.94e+08	0	4.51e+12	3.33e+08	23	9.01e+11	1.30e+08	100	1.8e+11
17	194	92610	3.43e+08	0	1.84e+12	2.22e+08	97	3.68e+11	5.80e+07	100	7.35e+10
18	194	97037	3.55e+08	0	1.93e+12	2.13e+08	87	3.85e+11	9.98e+07	100	7.7e+10

performs equally well in all instances through the diversity of solutions. Moreover, the mean of its runtime is significantly less in the half of instances, although the population size of Algorithm 21 can be significantly higher than the other two EAs.

Table 7.3. Comparison in ratio, number of required fitness evaluations and required CPU time for hitting the optimal value in 30 independent runs.

In.	n	QD				$(1+1)$EA				$(\mu+1)$EA			
		ratio	mean	time	Stat	ratio	mean	time	Stat	ratio	mean	time	Stat
1	50	100	1.53e+06	2.74e+01	2^-3^-	40	1.32e+09	4.49e+03	1^+3^*	40	4.59e+08	4.92e+03	1^+2^*
2	50	100	5.32e+05	8.14e+00	2^*3^*	100	5.30e+05	2.02e+00	1^*3^*	100	6.10e+05	6.21e+00	1^*2^*
3	50	100	2.43e+04	3.48e-01	2^+3^+	100	1.01e+04	4.38e-02	1^-3^*	100	1.21e+04	1.50e-01	1^-2^*
4	75	100	5.30e+06	8.28e+01	2^*3^*	97	1.20e+08	5.70e+02	1^*3^*	100	3.46e+07	6.90e+02	1^*2^*
5	75	100	3.63e+06	7.11e+01	2^*3^*	100	6.34e+07	3.44e+02	1^*3^*	100	3.16e+07	4.44e+02	1^*2^*
6	75	100	1.17e+05	2.21e+00	2^+3^+	100	1.30e+04	8.98e-02	1^-3^-	100	2.14e+04	3.20e-01	1^-2^+
7	100	100	2.42e+07	4.75e+02	2^-3^-	63	5.72e+08	3.26e+03	1^+3^*	43	2.51e+08	4.92e+03	1^+2^*
8	100	100	9.56e+06	2.02e+02	2^+3^+	100	2.33e+06	1.46e+01	1^-3^*	100	3.56e+06	5.81e+01	1^-2^*
9	100	100	6.18e+05	1.06e+01	2^+3^+	100	3.72e+04	2.24e-01	1^-3^*	100	5.03e+04	8.84e-01	1^-2^*
10	123	100	3.56e+07	6.65e+02	2^-3^-	77	3.90e+08	2.44e+03	1^+3^*	47	2.34e+08	4.70e+03	1^+2^*
11	123	100	2.05e+07	5.40e+02	2^*3^+	97	1.38e+08	1.13e+03	1^*3^*	87	6.10e+07	1.41e+03	1^-2^*
12	123	100	1.45e+06	3.74e+01	2^+3^+	100	6.55e+04	5.24e-01	1^-3^*	100	6.71e+04	1.34e+00	1^-2^*
13	151	100	5.04e+07	9.15e+02	2^*3^*	97	1.25e+08	1.07e+03	1^*3^*	87	8.65e+07	2.14e+03	1^*2^*
14	151	100	4.27e+07	9.75e+02	2^+3^+	100	1.20e+07	1.06e+02	1^-3^*	100	1.10e+07	3.33e+02	1^-2^*
15	151	100	3.18e+06	9.87e+01	2^+3^+	100	1.17e+05	1.09e+00	1^-3^*	100	1.09e+05	2.59e+00	1^-2^*
16	194	100	1.58e+08	4.22e+03	2^-3^-	57	4.99e+08	4.21e+03	1^+3^*	47	2.34e+08	5.47e+03	1^+2^*
17	194	100	1.18e+08	2.25e+03	2^-3^*	57	4.29e+08	3.91e+03	1^+3^*	40	2.07e+08	4.87e+03	1^*2^*
18	194	100	7.76e+06	1.50e+02	2^+3^+	100	1.17e+05	1.42e+00	1^-3^*	100	1.37e+05	4.71e+00	1^-2^*

7.5 Conclusions

In this chapter, we examined the capability of QD approaches and, in particular, MAP-Elite in solving KP. We defined two behavioural spaces inspired by the classic DP approaches and two corresponding MAP-Elite-based algorithms operating on these spaces. We established that they imitate the exact DP approach, and one of them behaves similarly to the classic FPTAS for KP under a specific parameter setting, making it a FPRAS. We then compared the runtime of the algorithms empirically on instances of various properties related to their hardness and found that the MAP-Elite selection mechanism significantly boosts the efficiency of EAs in solving KP in terms of convergence ratio, especially in hard instances. Inspecting the behavioural spaces and population sizes reveals that smaller populations correlate to faster optimisation, demonstrating a well-known trade-off between optimisation and exploring behavioural spaces.

It is an open question to which extent MAP-Elites can simulate DP-like behaviours in other problems with recursive subproblem structures. Moreover, it might be possible to make such approaches outperform DP via better controls of behavioural space exploration, combined with more powerful variation operators.

Chapter 8

Constructing Diverse Satisfying Assignments

8.1 Introduction

In previous chapters, we introduced EDO-based approaches to construct diverse sets of solutions for the TSP and the TTP. We employed variation operators in these EAs to generate offspring. Nevertheless, conventional variation operators have little chance to generate feasible offspring in heavily constrained optimisation problems. This chapter focuses on the diversification of assignments (solutions) for the Boolean satisfiability problem (SAT). Several characteristics distinguish SAT from other problems studied in the EDO literature and previous chapters. For instance, the other problems contain either no or few constraints, such as the KP and the TSP. SAT, however, is a highly constrained problem, making it extremely difficult to generate a feasible solution with conventional operators and algorithms in the literature of EDO. In other fields, such as constrained programming, researchers often forbid some variables or elements of a given problem to construct a diverse set of solutions. This paper makes a bridge between this approach and EDO.

Instead of using conventional operators, which are inefficient in SAT, we introduce evolutionary algorithms (EAs) and operators that iteratively modify the original SAT problem by adding clauses. We use a time-efficient solver, minisat [32], to construct new solutions and utilise EDO approaches to maximise the diversity of the solutions. It should be noted that standard minisat is a 20-year-old algorithm, but some variants of it are still competitive. Having said that, we believe the standard minisat is a beneficial starting point for this study.

We define two entropy-based diversity measures to quantify the diversity of SAT assignments. The first measure treats all variables equally, while the other takes the frequency of variables in clauses into account. We also conduct a comprehensive experimental investigation, the goal of which is twofold: First, to evaluate the algorithms' performance in constructing diverse assignments. And second, to study the correlation among diversity, solution space, and the number of clauses. For this purpose, we use an SAT generator to construct instances with particular characteristics. Then, we observe how the changes in these characteristics affect the diversity of solutions and algorithms' performances. For example, The introduced mutation outperforms the crossover in the power law SAT instances, while it is the opposite in the uniform instances.

The work of this chapter is based on a conference paper [89] presented at The genetic and evolutionary computation conference (GECCO 2023). The remainder of the chapter is structured as follows: We first define SAT and diversity in Section 8.2. The diversity algorithms are introduced in Section 8.3. The Comprehensive experimental investigation is presented in Section 8.4. Finally, we finish with concluding remarks.

8.2 SAT and Diversity

This chapter aims to compute a diverse set of assignments for a given formula. The SAT problem was defined in Section 2.4.4. For this purpose, we require a measure to quantify the diversity of assignments.

8.2.1 Diversity

We utilise an entropy-based measure of diversity. First, we define some notations. Let X denote the set of Boolean variables, $x = (x_1, \cdots, x_n)$ the assignment, and P a set of assignments, where $|X| = n$, $|P| = \mu$, m is number of the clauses. Also, let $f(x_i)$ be the number of assignments in P, where $x_i = True$. Then, we can calculate the contribution of each variable to diversity as

$$h(x_i) = \begin{cases} 0 & \text{if } f(x_i) = 0 \text{ and} \\ -\left(\frac{f(x_i)}{\mu}\right) \cdot \ln\left(\frac{f(x_i)}{\mu}\right) & \text{if } f(x_i) > 0. \end{cases}$$

In line with EDO literature [83, 85], the entropy of P can be calculated by summation of the variables' contributions:

$$H_1(P) = \sum_{x_i \in X} h(x_i)$$

Nevertheless, some variables appear in clauses more frequently than others. It would be intriguing to give such variables more weight in the entropy calculation, Therefore, we define the second measure as follows:

$$H_2(P) = \sum_{x_i \in X} r(x_i) \cdot h(x_i),$$

where $r(x_i)$ is the number of occurrences of x_i in the formula. It is beneficial to know the maximum diversity for the measures. It can be used as an upper bound to evaluate the diversity of a set of solutions. We can calculate the optimal $f(x)$ from $\frac{dh(x)}{df(x)} = 0$; Thus, the contribution of a variable is at maximum when:

$$f(x) = \mu \cdot e^{-1}$$

Let denote the optimal $f(x)$ by f^*. Since there are no limitations on the number of true variables in P , H_1 and H_2 are maximum when $\{f(x) = f^* | \forall x \in X\}$. Then, we can calculate H_1^{max} and H_2^{max} form :

$$H_1^{max} = n \cdot f^* \& H_2^{max} = C \cdot f^*$$

where C is the number of the literals in Φ.

8.3 Diversity Algorithms

In this chapter, we compute a diverse set of assignments for a given SAT problem using the well-known SAT solver minisat. A basic approach to compute P for an SAT problem is to forbid the current assignment by adding a clause to the formula and using the solver to generate another one. For constructing the clause, we can easily make a disjunction of the literals where each literal is the flipped associated variable in the assignment. This method only sometimes leads to a diverse set of assignments. Algorithm 23 outlines the steps required for this approach.

Algorithm 23 The basic algorithm

```
1: while $|P| < \mu$ do
2:     Solve the SAT problem by the solver.
3:     if A satisfying assignment $x$ was found then
4:         Add $x$ to $P$.
5:         Add a clause forbidding $x$ to $\Phi$.
6:     else
7:         Break.
```

EDO is another method to compute a diverse set of assignments. We can fix some variables to true or false and then use the solver (minisat) to determine a satisfying assignment with those fixed variables. Afterwards, we can employ EDO approaches to maximise diversity. Here, the question is how to choose the fixed variables. In line with most EDO algorithms in the literature, we can randomly select one of the current solutions and, by standard bit flip mutation, flip some of the variable assignments and fix them. In contrast to the standard bit-flip mutation, where the rest of the variables remain unchanged, the solver determines the value for the other variables. Algorithm 24 describes this approach. First, we find the first satisfying assignment for Φ by

Algorithm 24 The bit-flip evolutionary algorithm

```
1: Solve the SAT problem by the solver and add $x$ to $P$.
2: while A termination criterion is met do
3:     Select an assignment $x$ from $P$ uniformly at random.
4:     Select and flip each variable independently with probability $\frac{1}{n}$.
5:     Add clauses fixing the selected variables to $\Phi$
6:     Solve $\Phi$ and determine unfixed variables by the solver.
7:     if A satisfying assignment $x$ was found then
8:         if $|P| > \mu$ then
9:             Add $x$ to $P$.
10:            Remove one individual $x$ from $P$, where $x = \arg\max_{x \in P} H(P \setminus \{x\})$.
11:    Remove the clauses that fix the variables from $\Phi$.
```

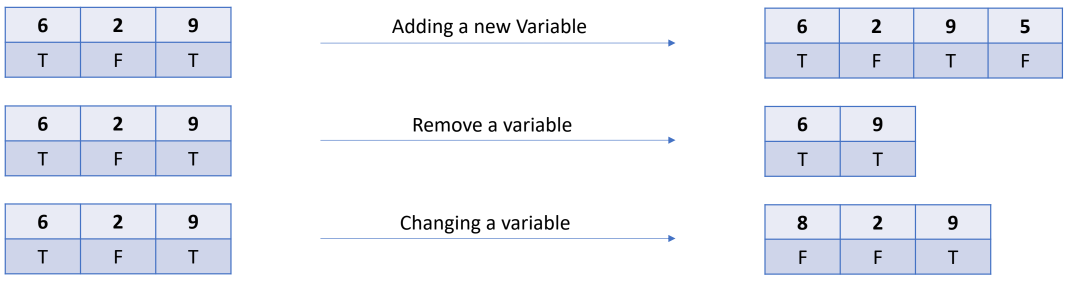
minisat and add it to P. Then, we select a solution in P uniformly at random and choose and flip some variables by the bit-flip mutation. After adding clauses to Φ that fix the selected variables, we solve Φ by minisat. If a satisfying assignment is found, we add it to P; Then, if $|P| > \mu$, we remove an assignment x with the least contribution to the diversity of P. Finally, we remove the clauses fixing the variables from Φ. We repeat these steps until a termination criterion is met.

Algorithm 25 The EDO algorithm

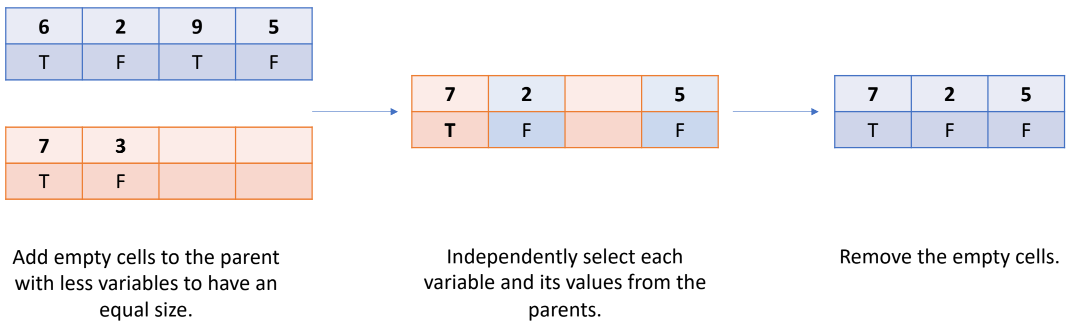
```
1: while $|P| < \mu$ do
2:     Randomly fix $l$ variables (determine $y$).
3:     Add the clauses that fix the variables in $y$ to $\Phi$ and solve it by the solver.
4:     if A satisfying assignment $x$ was found then
5:         Add $x$ to $P$ and $y$ to $Y$.
6:     Remove the clauses fixing the variables from $\Phi$.
7: while A termination criterion is met do
8:     Randomly select one (two) parent(s) $y_i$ ($y_j$) from $Y$.
9:     Generate a new solution $y_o$ by mutation or crossover + mutation.
10:    Add clauses that fix the variables in $y_o$ to $\Phi$ and solve the SAT problem.
11:    if A satisfying assignment $x$ is found then
12:        Add $x$ to $P$ and $y_o$ to $Y$.
13:        Remove one individual $x$ from $P$, where $x = \arg\max_{x \in P} H(P \setminus \{x\})$, and the
           corresponding solution $y$ from $Y$.
14:    Remove the clauses fixing the variables from $\Phi$.
```

Since minisat is an exact algorithm, we can map from the fixed variables to the actual assignments. Thus, we can save the fixed variables and operate (crossover, mutation) on them. So, we have a solution y consisting of a string $y' = (y'_1, \cdots, y'_l)$ showing the index of fixed variables and a Boolean string y'' showing their values. Let Y be a set of solutions y, where $|Y| = \mu$. Note that from each $y_i \in Y$ we can map to $x_i \in P$, by fixing variables in y_i and solving the problem by the solver.

Algorithm 25 sketches the steps required in this approach. The algorithm consists of two stages, the initialisation and the evolutionary stage. In initialisation, we randomly generate a variable y, where $|y| = l$. We solve ϕ after adding clauses to it.



(A) Mutation


6
2
9
5
T
F
T
F
7
3
T
F
7
2
5
T
F
F
7
2
5
T
F
F
Add empty cells to the parent with less variables to have an equal size.
Independently select each variable and its values from the parents.
Remove the empty cells.

(B) Crossover

FIGURE 8.1. The representation of solution y, the mutation, and the crossover in the EDO algorithm 25.

If a satisfying assignment x is found, we add x to P, and y to Y. Afterwards, we remove the clauses fixing the variables from Φ. we continue these steps until $|P| = \mu$.

TABLE 8.1. The diversity obtained from the algorithms using H_1 as the fitness function in 30 independent runs. Stat shows the results of the Kruskal-Wallis statistical test at a 5% significance level with Bonferroni correction. In row Stat, the notation X^+ means the median of the measure (H_1) is better than the one for variant X, X^- means it is worse, and X^* indicates no significant difference.

	Basic 23			Bit-flip 24			EDO 25 Mutation			EDO 25 Crossover+Mutation		
m	H_1	H_2	Stat (1)	H_1	H_2	Stat (2)	H_1	H_2	Stat (3)	H_1	H_2	Stat (4)
210	0.055	0.016	$2^-3^-4^-$	0.753	0.839	$1^+3^-4^-$	0.962	0.959	$1^+2^+4^*$	0.953	0.955	$1^+2^+3^*$
220	0.052	0.011	$2^-3^-4^-$	0.721	0.818	$1^+3^-4^-$	0.945	0.938	$1^+2^+4^*$	0.932	0.933	$1^+2^+3^*$
230	0.055	0.019	$2^-3^-4^-$	0.738	0.823	$1^+3^-4^-$	0.937	0.932	$1^+2^+4^*$	0.925	0.925	$1^+2^+3^*$
240	0.046	0.007	$2^-3^-4^-$	0.731	0.808	$1^+3^-4^-$	0.933	0.927	$1^+2^+4^*$	0.921	0.924	$1^+2^+3^*$
250	0.171	0.135	$2^-3^-4^-$	0.774	0.851	$1^+3^-4^-$	0.928	0.918	$1^+2^+4^*$	0.911	0.915	$1^+2^+3^*$
260	0.114	0.075	$2^-3^-4^-$	0.765	0.832	$1^+3^-4^-$	0.925	0.909	$1^+2^+4^*$	0.914	0.904	$1^+2^+3^*$
270	0.089	0.061	$2^-3^-4^-$	0.757	0.823	$1^+3^-4^-$	0.911	0.893	$1^+2^+4^*$	0.896	0.886	$1^+2^+3^*$
280	0.172	0.143	$2^-3^-4^-$	0.76	0.828	$1^+3^-4^-$	0.907	0.897	$1^+2^+4^*$	0.886	0.885	$1^+2^+3^*$
290	0.14	0.083	$2^-3^-4^-$	0.826	0.842	$1^+3^-4^-$	0.912	0.878	$1^+2^+4^+$	0.9	0.874	$1^+2^+3^-$
300	0.272	0.235	$2^-3^-4^-$	0.825	0.825	$1^+3^-4^-$	0.902	0.856	$1^+2^+4^*$	0.895	0.857	$1^+2^+3^*$
310	0.191	0.156	$2^-3^-4^-$	0.776	0.777	$1^+3^-4^*$	0.862	0.814	$1^+2^+4^*$	0.844	0.806	$1^+2^*3^*$
320	0.099	0.051	$2^-3^-4^-$	0.478	0.424	$1^+3^-4^*$	0.611	0.489	$1^+2^+4^*$	0.591	0.478	$1^+2^*3^*$
330	0.169	0.135	$2^-3^-4^-$	0.544	0.503	$1^+3^-4^*$	0.666	0.56	$1^+2^+4^*$	0.643	0.547	$1^+2^*3^*$
340	0.182	0.129	$2^-3^-4^-$	0.627	0.562	$1^+3^-4^-$	0.73	0.611	$1^+2^+4^*$	0.717	0.603	$1^+2^+3^*$
350	0.157	0.113	$2^-3^-4^-$	0.534	0.496	$1^+3^-4^*$	0.61	0.532	$1^+2^+4^*$	0.605	0.531	$1^+2^*3^*$
360	0.089	0.047	$2^-3^-4^-$	0.531	0.501	$1^+3^-4^*$	0.606	0.537	$1^+2^+4^*$	0.6	0.535	$1^+2^*3^*$
370	0.156	0.11	$2^-3^-4^-$	0.425	0.339	$1^+3^-4^-$	0.535	0.394	$1^+2^+4^*$	0.529	0.392	$1^+2^+3^*$
380	0.161	0.121	$2^-3^-4^-$	0.437	0.344	$1^+3^-4^*$	0.498	0.375	$1^+2^+4^*$	0.491	0.372	$1^+2^*3^*$

Table 8.2. The diversity obtained from the algorithms using H_1 as the fitness function. The variables appear in clauses based on a uniform distribution with $n = 100$ and $k = 3$. The notations are in line with Table 8.1

	Basic 23			Bit-flip 24			EDO 25 Mutation			EDO 25 Crossover+Mutation		
m	H_1	H_2	Stat (1)	H_1	H_2	Stat (2)	H_1	H_2	Stat (3)	H_1	H_2	Stat (4)
270	0.295	0.28	$2^{-}3^{-}4^{-}$	0.859	0.889	$1^{+}3^{-}4^{-}$	0.942	0.947	$1^{+}2^{+}4^{*}$	0.94	0.948	$1^{+}2^{+}3^{*}$
280	0.241	0.217	$2^{-}3^{-}4^{-}$	0.867	0.879	$1^{+}3^{-}4^{-}$	0.944	0.943	$1^{+}2^{+}4^{*}$	0.944	0.946	$1^{+}2^{+}3^{*}$
290	0.202	0.186	$2^{-}3^{-}4^{-}$	0.834	0.848	$1^{+}3^{-}4^{-}$	0.937	0.938	$1^{+}2^{+}4^{*}$	0.939	0.941	$1^{+}2^{+}3^{*}$
300	0.183	0.175	$2^{-}3^{-}4^{-}$	0.877	0.888	$1^{+}3^{-}4^{-}$	0.943	0.943	$1^{+}2^{+}4^{*}$	0.946	0.946	$1^{+}2^{+}3^{*}$
310	0.09	0.078	$2^{-}3^{-}4^{-}$	0.875	0.893	$1^{+}3^{-}4^{-}$	0.943	0.946	$1^{+}2^{+}4^{*}$	0.945	0.948	$1^{+}2^{+}3^{*}$
320	0.062	0.051	$2^{-}3^{-}4^{-}$	0.884	0.894	$1^{+}3^{-}4^{-}$	0.936	0.939	$1^{+}2^{+}4^{*}$	0.937	0.94	$1^{+}2^{+}3^{*}$
330	0.157	0.137	$2^{-}3^{-}4^{-}$	0.885	0.895	$1^{+}3^{-}4^{-}$	0.927	0.927	$1^{+}2^{+}4^{*}$	0.932	0.934	$1^{+}2^{+}3^{*}$
340	0.135	0.117	$2^{-}3^{-}4^{-}$	0.898	0.905	$1^{+}3^{-}4^{-}$	0.928	0.927	$1^{+}2^{+}4^{*}$	0.933	0.933	$1^{+}2^{+}3^{*}$
350	0.073	0.062	$2^{-}3^{-}4^{-}$	0.895	0.903	$1^{+}3^{-}4^{-}$	0.916	0.918	$1^{+}2^{+}4^{*}$	0.918	0.92	$1^{+}2^{+}3^{*}$
360	0.08	0.067	$2^{-}3^{-}4^{-}$	0.866	0.875	$1^{+}3^{-}4^{-}$	0.893	0.896	$1^{+}2^{+}4^{*}$	0.898	0.903	$1^{+}2^{+}3^{*}$
370	0.084	0.07	$2^{-}3^{-}4^{-}$	0.851	0.862	$1^{+}3^{-}4^{-}$	0.884	0.886	$1^{+}2^{+}4^{*}$	0.891	0.895	$1^{+}2^{+}3^{*}$
380	0.058	0.042	$2^{-}3^{-}4^{-}$	0.846	0.855	$1^{+}3^{-}4^{-}$	0.876	0.879	$1^{+}2^{+}4^{*}$	0.877	0.88	$1^{+}2^{+}3^{*}$
390	0.178	0.178	$2^{-}3^{-}4^{-}$	0.822	0.822	$1^{+}3^{*}4^{-}$	0.832	0.829	$1^{+}2^{*}4^{*}$	0.835	0.832	$1^{+}2^{+}3^{*}$
400	0.226	0.215	$2^{-}3^{-}4^{-}$	0.637	0.622	$1^{+}3^{-}4^{-}$	0.648	0.63	$1^{+}2^{+}4^{*}$	0.647	0.629	$1^{+}2^{+}3^{*}$
410	0.105	0.098	$2^{-}3^{-}4^{-}$	0.674	0.669	$1^{+}3^{-}4^{-}$	0.693	0.685	$1^{+}2^{+}4^{*}$	0.693	0.684	$1^{+}2^{+}3^{*}$
420	0.125	0.118	$2^{-}3^{-}4^{-}$	0.603	0.592	$1^{+}3^{*}4^{-}$	0.612	0.599	$1^{+}2^{*}4^{*}$	0.613	0.6	$1^{+}2^{+}3^{*}$
430	0.153	0.146	$2^{-}3^{-}4^{-}$	0.311	0.299	$1^{+}3^{*}4^{-}$	0.326	0.309	$1^{+}2^{*}4^{*}$	0.326	0.309	$1^{+}2^{+}3^{*}$
440	0.059	0.047	$2^{-}3^{-}4^{-}$	0.352	0.335	$1^{+}3^{-}4^{-}$	0.366	0.346	$1^{+}2^{+}4^{*}$	0.366	0.347	$1^{+}2^{+}3^{*}$

Having constructed an initial population, we move to the evolutionary stage. We first select a solution y (or two solutions in case of crossover) from Y and generate an offspring y_o by mutation (or first crossover, then mutation). After adding clauses fixing variables in y_o to Φ, we solve it by the solver. If a satisfying assignment x is found, we add x to P and y to Y; then remove a x from P and the corresponding y from Y that has the least contribution to the diversity of P. In the last step, we remove the clauses fixing the variables from Φ. We repeat these steps in the evolutionary stage until a termination criterion is met.

We now describe the operators, the mutation and the crossover. For the mutation, we take one of the following three actions uniformly at random: 1) Fix another variable (add a new variable to y), 2) unfix a variable (remove a variable from y), or 3) switch a fixed variable with an unfixed variable, all uniformly at random. The steps are depicted in Figure8.1a. Turning to the crossover, we add empty cells to the parent with fewer fixed variables to make the sizes equal. Then, we select each variable randomly from the parents with probability 1/2. Figure8.1b illustrates the steps required by the crossover.

8.4 Experimental Investigation

This section empirically studies and compares the introduced algorithms. We examine two variations of Algorithm 25: One solely employs mutation as the operator, while the other first generates an offspring by crossover and then uses mutation on the offspring. To examine the algorithms, we use the SAT generator [5] to generate two sets of CNF formulas. The SAT generator is also used for experimental investigations in [38, 39]. In the first set, the variables appear in clauses based on a power law distribution. The following parameters were used in generating the first set: $n = 100$, $k = 3$, $\beta = 2.75$, and $m = \{210, 220, \cdots, 380\}$, where k and β are the number of literals in a clause and the power law exponent, respectively. In the second set, the variables appear in the clauses based on the uniform distribution. The parameters for the set are: $n = 100$, $k = 3$, and $m = \{270, 280, \cdots, 440\}$. We set $\mu = 20$ and consider 2000 iterations as the termination criterion for the EAs. Instead of 30 independent runs on one formula, we generate 30 formulas for each configuration and run the algorithms once on each formula. This helps us to comprehend more about SAT instances having the same characteristics. Note that we made sure all formulas were satisfiable ($\Phi = true$).

8.4.1 Comparison of algorithms employing H_1 as the fitness

In this section, we compare the diversity of SAT assignments obtained by the presented algorithms using H_1 as the fitness function. Table 8.1 summarises the algorithms' results in the first set of instances (formulas). As expected, the basic algorithm results in assignments with poor diversity; the H_1 values range between 1.71 and 10.01. If we normalise these values, the range is from 5% to 27%. The interesting information is that the increase in the clause-variable ratio $\frac{m}{n}$ has no meaningful impact on the basic algorithm's result. The expectation is that an increase in $\left(\frac{m}{n}\right)$ reduces the feasible region, which leads to a decrease in the diversity of assignments; we can observe the trend in the results of the other algorithms.

As Table 8.1 shows, the bit-flip brings about considerably more diverse assignments than the basic algorithm. The observation can be confirmed by the Kruskal-Wallis statistical test at a 5% significance level and with Bonferroni correction. The mean of diversity ranges from 44% to 82%. Although there are also fluctuations in the bit-flip algorithm's results, we can observe a general decrease in diversity by an increase in $\left(\frac{m}{n}\right)$, especially when m is larger than 290. However, if we only consider the first half of the table, it is exactly the other way around; there is a slight increase in diversity obtained. One plausible reason is that the minisat solver is an exact algorithm, and bit-flip mutation does not impose as significant changes as required. On the other hand, an increase in $\left(\frac{m}{n}\right)$ makes even minor changes significantly impact the assignments. In fact, the feasible regain, and the maximum achievable diversity decrease in instances with medium values of $\left(\frac{m}{n}\right)$ compared to small ones, but the bit-flip algorithm performs better in these instances.

Table 8.1 indicates the superiority of EDO algorithms in constructing diverse sets of SAT assignments. Both algorithm variants yield decent results and statistically outperform the basic and the bit-flip algorithms in all instances. Here, we can observe a more static downward trend in diversity with increasing $\frac{m}{n}$. It results in sets with more than 90% diversity (normalised H_1) for instances with $\frac{m}{n} \leq 3$. For example, the mean of diversity is 96% in cases where $m = 210$. Interestingly, the variant using only mutation results in slightly higher diversity. Although, it is not statistically significant.

Table 8.2 draws a similar comparison between the algorithms on the set of uniform formulas. Almost all our observations in Table 8.1 are still valid. Table 8.2 shows that: 1) Algorithm 23 results in solutions with poor diversity ranging from 6% to 29%. Nevertheless, the diversity obtained in the uniform instances is higher compared to the power law formulas. 2) Bit-flip performs better than the basic algorithm but worse than the EDO variants. The average H_1 obtained by the bit-flip algorithms ranges from 0.31 to 0.86. 3) We can observe a descending trend in diversity for increasing $\frac{m}{n}$, especially in the EDO algorithms' results.

The most interesting part of the table is comparing the two EDO variants. In contrast to the power law instances, the variant using both crossover and mutation slightly outperforms the other one in terms of H_1. We can get diverse sets of SAT assignments with more than 90% diversity in terms of H_1 with the EDO algorithm in cases $m \leq 360$.

TABLE 8.3. The diversity obtained from the algorithms using H_2 as the fitness function on the same instances in Table 8.1. The Kruskal-Wallis statistical test is conducted on H_2. The notations are in line with Table 8.1

	Basic [23]			Bit-flip [24]			EDO [25] Mutation			EDO [25] Crossover+Mutation		
m	H_2	H_1	Stat (1)	H_2	H_1	Stat (2)	H_2	H_1	Stat (3)	H_2	H_1	Stat (4)
210	0.016	0.055	$2^-3^-4^-$	0.849	0.732	$1^+3^-4^-$	0.965	0.944	$1^+2^+4^*$	0.957	0.912	$1^+2^+3^*$
220	0.011	0.052	$2^-3^-4^-$	0.83	0.709	$1^+3^-4^-$	0.95	0.928	$1^+2^+4^*$	0.939	0.892	$1^+2^+3^*$
230	0.019	0.055	$2^-3^-4^-$	0.836	0.719	$1^+3^-4^-$	0.941	0.911	$1^+2^+4^+$	0.932	0.882	$1^+2^+3^-$
240	0.007	0.046	$2^-3^-4^-$	0.825	0.706	$1^+3^-4^-$	0.936	0.896	$1^+2^+4^*$	0.929	0.874	$1^+2^+3^*$
250	0.135	0.171	$2^-3^-4^-$	0.859	0.757	$1^+3^-4^-$	0.927	0.909	$1^+2^+4^*$	0.918	0.874	$1^+2^+3^*$
260	0.075	0.114	$2^-3^-4^-$	0.845	0.751	$1^+3^-4^-$	0.919	0.906	$1^+2^+4^*$	0.911	0.872	$1^+2^+3^*$
270	0.061	0.089	$2^-3^-4^-$	0.838	0.737	$1^+3^-4^-$	0.907	0.89	$1^+2^+4^+$	0.895	0.844	$1^+2^+3^-$
280	0.143	0.172	$2^-3^-4^-$	0.842	0.74	$1^+3^-4^-$	0.906	0.877	$1^+2^+4^*$	0.895	0.84	$1^+2^+3^*$
290	0.083	0.14	$2^-3^-4^-$	0.861	0.807	$1^+3^-4^-$	0.895	0.887	$1^+2^+4^+$	0.887	0.864	$1^+2^+3^-$
300	0.235	0.272	$2^-3^-4^-$	0.835	0.813	$1^+3^-4^-$	0.865	0.884	$1^+2^+4^*$	0.865	0.867	$1^+2^+3^*$
310	0.156	0.191	$2^-3^-4^-$	0.786	0.762	$1^+3^-4^*$	0.824	0.845	$1^+2^+4^*$	0.816	0.815	$1^+2^*3^*$
320	0.051	0.099	$2^-3^-4^-$	0.434	0.468	$1^+3^-4^*$	0.494	0.599	$1^+2^+4^*$	0.481	0.558	$1^+2^*3^*$
330	0.135	0.169	$2^-3^-4^-$	0.513	0.534	$1^+3^-4^*$	0.562	0.647	$1^+2^+4^*$	0.551	0.61	$1^+2^*3^*$
340	0.129	0.182	$2^-3^-4^-$	0.57	0.617	$1^+3^-4^*$	0.616	0.718	$1^+2^+4^*$	0.606	0.69	$1^+2^*3^*$
350	0.113	0.157	$2^-3^-4^-$	0.505	0.524	$1^+3^-4^*$	0.537	0.598	$1^+2^+4^*$	0.534	0.587	$1^+2^*3^*$
360	0.047	0.089	$2^-3^-4^-$	0.504	0.524	$1^+3^-4^*$	0.541	0.602	$1^+2^+4^*$	0.536	0.589	$1^+2^*3^*$
370	0.11	0.156	$2^-3^-4^-$	0.342	0.42	$1^+3^-4^-$	0.396	0.53	$1^+2^+4^*$	0.392	0.521	$1^+2^+3^*$
380	0.121	0.161	$2^-3^-4^-$	0.347	0.432	$1^+3^-4^*$	0.377	0.494	$1^+2^+4^*$	0.374	0.484	$1^+2^*3^*$

Table 8.4. The diversity obtained from the algorithms using H_2 the fitness on the same instances in Table 8.2. The notations are in line with Table 8.3.

	Basic 23			Bit-flip 24			EDO 25 Mutation			EDO 25 Crossover+Mutation		
m	H_2	H_1	Stat (1)	H_2	H_1	Stat (2)	H_2	H_1	Stat (3)	H_2	H_1	Stat (4)
270	0.28	0.295	$2^{-}3^{-}4^{-}$	0.891	0.849	$1^{+}3^{-}4^{-}$	0.95	0.933	$1^{+}2^{+}4^{*}$	0.946	0.923	$1^{+}2^{+}3^{*}$
280	0.217	0.241	$2^{-}3^{-}4^{-}$	0.881	0.863	$1^{+}3^{-}4^{-}$	0.946	0.937	$1^{+}2^{+}4^{*}$	0.947	0.934	$1^{+}2^{+}3^{*}$
290	0.186	0.202	$2^{-}3^{-}4^{-}$	0.849	0.83	$1^{+}3^{-}4^{-}$	0.939	0.931	$1^{+}2^{+}4^{*}$	0.942	0.93	$1^{+}2^{+}3^{*}$
300	0.175	0.183	$2^{-}3^{-}4^{-}$	0.891	0.871	$1^{+}3^{-}4^{-}$	0.945	0.939	$1^{+}2^{+}4^{*}$	0.947	0.936	$1^{+}2^{+}3^{*}$
310	0.078	0.09	$2^{-}3^{-}4^{-}$	0.897	0.874	$1^{+}3^{-}4^{-}$	0.945	0.933	$1^{+}2^{+}4^{*}$	0.95	0.937	$1^{+}2^{+}3^{*}$
320	0.051	0.062	$2^{-}3^{-}4^{-}$	0.895	0.881	$1^{+}3^{-}4^{-}$	0.938	0.929	$1^{+}2^{+}4^{*}$	0.94	0.928	$1^{+}2^{+}3^{*}$
330	0.136	0.157	$2^{-}3^{-}4^{-}$	0.897	0.882	$1^{+}3^{-}4^{-}$	0.93	0.923	$1^{+}2^{+}4^{*}$	0.934	0.923	$1^{+}2^{+}3^{*}$
340	0.117	0.135	$2^{-}3^{-}4^{-}$	0.907	0.895	$1^{+}3^{-}4^{-}$	0.932	0.927	$1^{+}2^{+}4^{*}$	0.934	0.925	$1^{+}2^{+}3^{*}$
350	0.062	0.073	$2^{-}3^{-}4^{-}$	0.904	0.892	$1^{+}3^{-}4^{-}$	0.919	0.911	$1^{+}2^{+}4^{*}$	0.922	0.913	$1^{+}2^{+}3^{*}$
360	0.067	0.08	$2^{-}3^{-}4^{-}$	0.879	0.865	$1^{+}3^{-}4^{-}$	0.897	0.889	$1^{+}2^{+}4^{*}$	0.904	0.894	$1^{+}2^{+}3^{*}$
370	0.07	0.084	$2^{-}3^{-}4^{-}$	0.866	0.851	$1^{+}3^{-}4^{-}$	0.892	0.882	$1^{+}2^{+}4^{*}$	0.896	0.884	$1^{+}2^{+}3^{*}$
380	0.042	0.058	$2^{-}3^{-}4^{-}$	0.858	0.843	$1^{+}3^{-}4^{-}$	0.878	0.867	$1^{+}2^{+}4^{*}$	0.881	0.869	$1^{+}2^{+}3^{*}$
390	0.178	0.178	$2^{-}3^{-}4^{-}$	0.824	0.818	$1^{+}3^{*}4^{-}$	0.832	0.829	$1^{+}2^{*}4^{*}$	0.834	0.83	$1^{+}2^{+}3^{*}$
400	0.214	0.226	$2^{-}3^{-}4^{-}$	0.623	0.636	$1^{+}3^{-}4^{-}$	0.631	0.646	$1^{+}2^{+}4^{*}$	0.631	0.645	$1^{+}2^{+}3^{*}$
410	0.098	0.105	$2^{-}3^{-}4^{-}$	0.672	0.673	$1^{+}3^{-}4^{-}$	0.684	0.688	$1^{+}2^{+}4^{*}$	0.684	0.686	$1^{+}2^{+}3^{*}$
420	0.118	0.125	$2^{-}3^{-}4^{-}$	0.593	0.601	$1^{+}3^{-}4^{*}$	0.602	0.611	$1^{+}2^{+}4^{*}$	0.601	0.61	$1^{+}2^{*}3^{*}$
430	0.146	0.153	$2^{-}3^{-}4^{-}$	0.3	0.311	$1^{+}3^{*}4^{-}$	0.309	0.325	$1^{+}2^{*}4^{*}$	0.309	0.326	$1^{+}2^{+}3^{*}$
440	0.047	0.059	$2^{-}3^{-}4^{-}$	0.336	0.352	$1^{+}3^{-}4^{-}$	0.347	0.366	$1^{+}2^{+}4^{*}$	0.347	0.366	$1^{+}2^{+}3^{*}$

8.4.2 Comparison of algorithms employing H_2 as the fitness

We examine the algorithms' performance when H_2 is incorporated as the fitness function. The H_2 differs from H_1 in focusing on the variables with more appearances in Φ. Table 8.3 and 8.4 summarise the algorithms' results in the power law and uniform instances, respectively. Since Algorithm 24 does not use any diversity measures inside of the algorithm, the results are the same as those of Table 8.1 and 8.2. Nevertheless, the other algorithms' results in Table 8.3 and 8.4 are different to those in Table 8.1 and 8.2. As expected, the diversity of assignments slightly increases in terms of H_2, while there is a minor drop in H_1 values. The change is plausible since we incorporated H_2 into the algorithms as the fitness function instead of H_1.

One may observe that increasing $\frac{m}{n}$ affects the capability of the introduced algorithms in terms of H_2 more than it does in terms of H_1. This is because, in a limited feasible region, the more frequent variables are more likely to be fixed at true or false. Since those variables have a higher weight in the diversity calculation, increases in $\frac{m}{n}$ make it challenging to diversify solutions in terms of H_2. For instance, Table 8.3 indicates that the H_2 values drop from 0.96 to 0.38 for the EDO algorithm using mutation, while the same sets of solutions result in less severe decreases in H_1 values (from 0.94 to 0.5).

Table 8.3 also indicates that the gap between the results of the EDO algorithms' variants is more profound when H_2 is used as the fitness function. The statistical tests also confirm the difference in favour of the variant employing the mutation in instances where $m = \{230, 270, 290\}$. However, it is the other way around in the uniform instances; the variant that benefits from the crossover performs slightly better, although the difference is statically insignificant. The same observation we had when H_1 was incorporated into the algorithm as the fitness function. The results show the algorithms perform similarly in cases of using H_1 or H_2 measures. However, H_1 and H_2 are different since H_2 is biased toward the more frequent literals. Since the algorithms perform similarly, decision-makers can take into account their preferences and needs and choose one of the measures.

8.4.3 Investigation on Unsatisfiablity

This subsection studies the correlation between the obtained diversity and the number of unsatisfiable formulas generated during the search. The introduced algorithms, as mentioned, modify the formula Φ to generate a new assignment in each iteration. Although Φ is a true formula, it is likely to make it false via modifications during the search. We consider Algorithm 24 for this purpose since the algorithm does not have any hyper-parameters affecting the results.

Figure 8.2 depicts the trajectories of diversity and the false Φ generated by Algorithm 24. Note that we normalise the values to plot them in a figure. As expected,

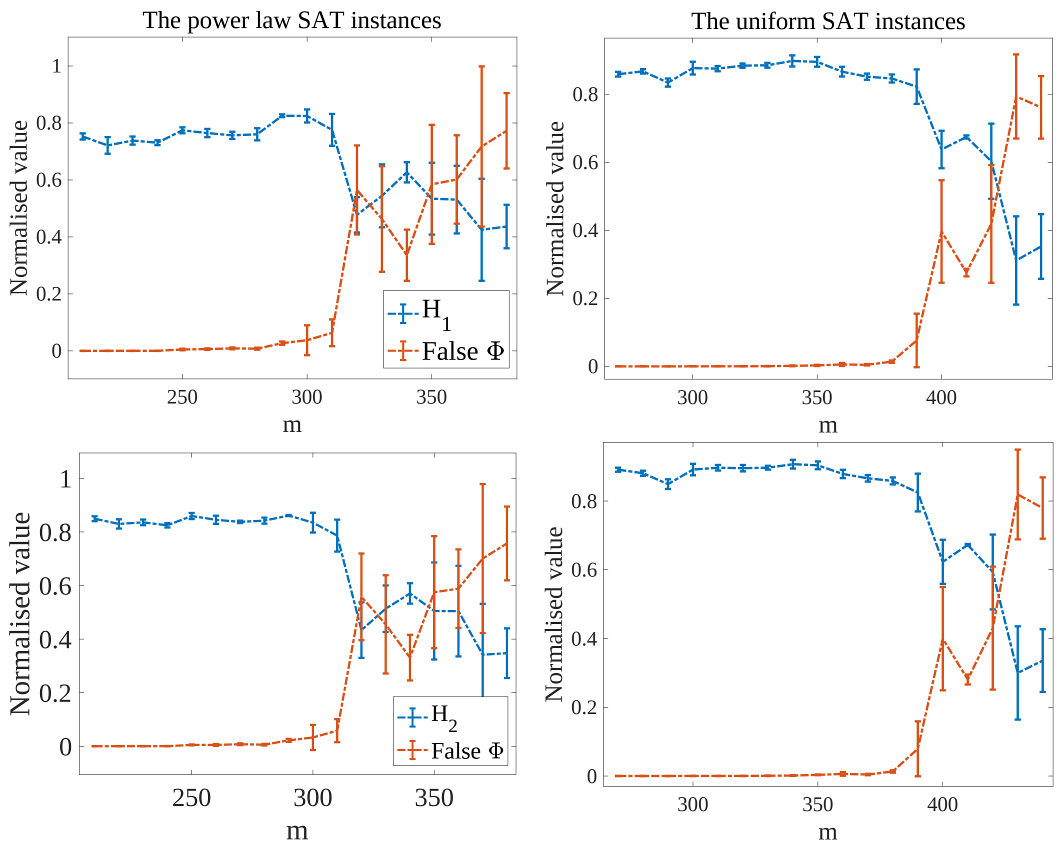


FIGURE 8.2. The representative trajectories of the bit-flip algorithm's diversity and the number of false Φ. In the first row, H_1 serves as the fitness function, while it is H_2 in the second row.

the algorithm generates the minimum number of false formulas (false Φ) when $\frac{m}{n}$ is low. Low values of $\frac{m}{n}$ often lead to large feasible regions and, consequently, a larger room to diversify the solutions. In such cases, the modifications of Algorithm 24 are not large enough to cause unsatisfiability for Φ. If $\frac{m}{n}$ gets sufficiently large, so does the feasible region get more limited, affecting both the diversity and satisfiability rate. Although a disproportional relationship between diversity and unsatisfiability is expected, the figure interestingly depicts a symmetric behaviour. The trajectories are pretty similar for H_1 and H_2. The sole difference is the range of H_1 and H_2 in the power law instances, where H_2 starts and finishes at slightly higher values.

8.5 Conclusion

This study presented evolutionary approaches to construct a diverse set of solutions in SAT using the well-known SAT solver, minisat. We first defined two measures to quantify the diversity of solutions. One, which considers and one, which dismisses the frequency of variable appearances in clauses. Then, we introduced two EAs, employing the EDO principle to construct a diverse set of SAT assignments. The EAs iteratively make modifications on a given SAT instance, then solve it with a well-known solver, minisat. Finally, we conducted a comprehensive experimental investigation to assess the algorithms' performance and study the solution space and unsatisfiability rate. The results indicate the capability of the introduced algorithm to compute highly diverse sets of SAT solutions.

For future studies, it is intriguing to study more complicated EAs, like $(\mu + \lambda)$-EAs. Although it is challenging in diversity problems to select the next generation when λ is larger than one, an increase in λ can potentially improve the algorithms' performance. Another possible extension is to study other related problems, such as MaxSAT.

Chapter 9

Conclusion

This thesis focuses on diversity optimisation in combinatorial problems. Firstly, we introduced an entropy-based diversity measure in the context of EDO. The maximum achievable entropy in the TSP was theoretically investigated. Then, MIP formulation was provided to compute a diverse set of TSP tours to support the theoretical proof for the maximum achievable entropy. Additionally, we proposed an $(\mu+1)$EA for the diversity problem in the TSP and defined a biased version of the 2-OPT operator. In the biased 2-OPT, the frequency of segments of the solution in the population plays a crucial role in the edge selection for mutation. Methods for selecting the next generation in $(\mu + \lambda)$EA, where $\lambda \geq 2$, were also introduced. Then, we conducted comprehensive experimental investigations to examine the performance of the proposed methods. The results indicate that biased 2-OPT results in faster algorithm convergence, and the EA-based selection method outperforms the vanilla $(\mu + 1)$EA in the literature.

In Chapter 4, we extended our approach in Chapter 3 by introducing EAX-EDO, which is a modified version of the EAX crossover to consider the quality and diversity of solutions simultaneously. This modification makes a bias toward maximising the diversity of TSP tours, while the vanilla EAX only focuses on the quality of solutions. Moreover, we investigated scenarios where the optimal solution may be unknown. Then, we compared our approaches with vanilla GA based on the EAX. The results demonstrate our method's ability to diversify TSP solutions.

After studying TSP within the context of EDO, we shifted our focus to another well-known combinatorial problem, the TTP in n Chapter 5. We first introduced a bi-level QD-based algorithm to address the TTP. For this purpose, we defined a 2D behavioural space based on the sub-problems of the TTP, meaning the TSP and the KP. Next, we incorporated a selection procedure based on MAP-Elite into the EA so as to explore niches in the TSP/KP behavioural space. We also empirically investigated the performance of the proposed algorithm. The results showed the strength of the algorithm in finding and diversifying the TTP solutions with decent objective values. Furthermore, in Chapter 6, we investigated the interdependency of TTP's sub-problems in terms of diversity using EDO. We also proposed a Co-EA to simultaneously compute diverse sets of solutions in both the behavioural and structural spaces.

In addition to our empirical study of QD, we also examined the concept from a theoretical perspective. This study presents the first runtime analysis of QD. We focused on the KP for the case study. We defined two behavioural spaces for the KP, a weight-based space and a profit-based space. Furthermore, we conducted comprehensive experimental analyses, indicating that the weight-based algorithm outperforms the profit-based one. The results also showed the weight-based algorithm is more likely to escape local optima than the proposed similar approaches, the $(1+1)$EA and $(\mu+1)$EA.

In the final technical chapter, we investigated a heavily constrained problem, the SAT. A large number of constraints in a problem make conventional variation operators less successful in generating feasible assignments. To address this, we exploit a well-known SAT solver by incorporating it into two different EAs. The EAs iteratively modify the problem itself by adding new constraints. These modifications enforce the SAT solver to construct distinct solutions. Similar to the other chapters, we examined the performance of the introduced EA through a series of experimental investigations, which shows that the EDO-based algorithm can construct highly diverse sets of assignments.

9.1 Future Studies

We have already provided future study suggestions related to each chapter at the end of each chapter, offering directions for further research. This section aims to introduce additional ways to extend this study more fundamentally. Diversity optimisation in bio-inspired computation is a rapidly evolving field of research, and this thesis primarily focuses on evolutionary-based algorithms. There are several other bio-inspired and metaheuristic algorithms that remain to be studied in this field, such as ACO and PSO. Conducting a comparison between these algorithms and the introduced EAs in this study would provide a comprehensive understanding of their performance in diversity optimisation.

Furthermore, our concentration was primarily on a selection of combinatorial optimisation problems, including TSP, TTP, and KP. However, there are numerous benchmark optimization problems of significant importance in both continuous and discrete domains. For instance, the vehicle routing problem (VRP) is another vital multi-component combinatorial optimization problem with diverse real-world applications. In recent years, sustainability has been incorporated into VRP, giving rise to the concept of green VRP. Most solution approaches in green VRP involve multi-objective frameworks such as multi-objective EAs. It would be intriguing to study such problems in the context of QD by defining a behavioural space considering sustainable features.

In diversity optimisation, EDO and QD aim to find a diverse set of solutions for a given optimisation problem. Therefore, we can treat the population as a singular solution to the diversity problem such that we can investigate variation operators, such as mutation and crossover, on multiple populations. This unique approach allows us to simultaneously benefit from QD, EDO, and even niching and optimise diversity and quality. Finally, this thesis studied a number of well-known single objective optimisation problems, while there are many real-world problems, including several objective functions. It would be intriguing to study such problems in the context of EDO and QD.

Bibliography

[1] Emile HL Aarts and Jan Karel Lenstra. *Local search in combinatorial optimization.* Princeton University Press, 2003.

[2] Bradley Alexander, James Kortman, and Aneta Neumann. "Evolution of artistic image variants through feature based diversity optimisation". In: *GECCO.* ACM, 2017, pp. 171–178.

[3] Maxime Allard, Simón C. Smith, Konstantinos I. Chatzilygeroudis, and Antoine Cully. "Hierarchical quality-diversity for online damage recovery". In: *GECCO.* ACM, 2022, pp. 58–67.

[4] RD Angel, WL Caudle, R Noonan, and ANDA Whinston. "Computer-assisted school bus scheduling". In: *Management Science* 18.6 (1972), B–279.

[5] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. "Towards Industrial-Like Random SAT Instances". In: *IJCAI.* 2009, pp. 387–392.

[6] David L. Applegate, Robert E. Bixby, Vasek Chvátal, and William J. Cook. "Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems". In: *Math. Program.* 97.1-2 (2003), pp. 91–153.

[7] Gerardo Beni and Jing Wang. "Swarm intelligence in cellular robotic systems". In: *Robots and biological systems: towards a new bionics?* Springer. 1993, pp. 703–712.

[8] Mohammad Reza Bonyadi and Zbigniew Michalewicz. "Particle Swarm Optimization for Single Objective Continuous Space Problems: A Review". In: *Evol. Comput.* 25.1 (2017), pp. 1–54.

[9] Mohammad Reza Bonyadi, Zbigniew Michalewicz, and Luigi Barone. "The travelling thief problem: The first step in the transition from theoretical problems to realistic problems". In: *IEEE Congress on Evolutionary Computation.* IEEE, 2013, pp. 1037–1044.

[10] Mohammad Reza Bonyadi, Zbigniew Michalewicz, Michal Roman Przybylek, and Adam Wierzbicki. "Socially inspired algorithms for the travelling thief problem". In: *GECCO.* ACM, 2014, pp. 421–428.

[11] Jakob Bossek, Pascal Kerschke, Aneta Neumann, Markus Wagner, Frank Neumann, and Heike Trautmann. "Evolving diverse TSP instances by means of novel and creative mutation operators". In: *FOGA.* ACM, 2019, pp. 58–71.

[12] Jakob Bossek and Frank Neumann. "Evolutionary diversity optimization and the minimum spanning tree problem". In: *GECCO.* ACM, 2021, pp. 198–206.

[13] Jakob Bossek and Frank Neumann. “Exploring the feature space of TSP instances using quality diversity”. In: *GECCO*. ACM, 2022, pp. 186–194.

[14] Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2014.

[15] Jonatas B. C. Chagas and Markus Wagner. “A weighted-sum method for solving the bi-objective traveling thief problem”. In: *Comput. Oper. Res.* 138 (2022), p. 105560.

[16] Konstantinos I. Chatzilygeroudis, Antoine Cully, Vassilis Vassiliades, and Jean-Baptiste Mouret. “Quality-Diversity Optimization: a novel branch of stochastic optimization”. In: *CoRR* abs/2012.04322 (2020).

[17] Jeff Clune, Jean-Baptiste Mouret, and Hod Lipson. “Summary of "the evolutionary origins of modularity"”. In: *GECCO (Companion)*. ACM, 2013, pp. 23–24.

[18] G. V. Conroy. “*Handbook of genetic algorithms* by Lawrence Davis (Ed.), Chapman & Hall, London, 1991, pp 385, £32.50”. In: *Knowl. Eng. Rev.* 6.4 (1991), pp. 363–365.

[19] IBM ILOG Cplex. “V12. 1: User’s Manual for CPLEX”. In: *International Business Machines Corporation* 46.53 (2009), p. 157.

[20] Georges A Croes. “A method for solving traveling-salesman problems”. In: *Operations research* 6.6 (1958), pp. 791–812.

[21] Antoine Cully and Jean-Baptiste Mouret. “Behavioral repertoire learning in robotics”. In: *GECCO*. ACM, 2013, pp. 175–182.

[22] George Dantzig. “Discrete-variable extremum problems”. In: *Operations research* 5.2 (1957), pp. 266–288.

[23] George Dantzig. *Linear programming and extensions*. Princeton University Press, 2016.

[24] Anh Viet Do, Jakob Bossek, Aneta Neumann, and Frank Neumann. “Evolving diverse sets of tours for the travelling salesperson problem”. In: *GECCO*. ACM, 2020, pp. 681–689.

[25] Anh Viet Do, Mingyu Guo, Aneta Neumann, and Frank Neumann. “Analysis of Evolutionary Diversity Optimization for Permutation Problems”. In: *ACM Trans. Evol. Learn. Optim.* 2.3 (2022), 11:1–11:27.

[26] Benjamin Doerr and Carola Doerr. “Optimal parameter choices through self-adjustment: Applying the 1/5-th rule in discrete settings”. In: *GECCO Companion*. 2015, pp. 1335–1342.

[27] Benjamin Doerr, Carola Doerr, Aneta Neumann, Frank Neumann, and Andrew M. Sutton. “Optimization of Chance-Constrained Submodular Functions”. In: *AAAI*. AAAI Press, 2020, pp. 1460–1467.

[28] Benjamin Doerr, Anton V. Eremeev, Frank Neumann, Madeleine Theile, and Christian Thyssen. “Evolutionary algorithms and dynamic programming”. In: *Theor. Comput. Sci.* 412.43 (2011), pp. 6020–6035.

[29] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. "Fast genetic algorithms". In: *GECCO*. ACM, 2017, pp. 777–784.

[30] Benjamin Doerr and Frank Neumann, eds. *Theory of Evolutionary Computation - Recent Developments in Discrete Optimization*. Natural Computing Series. Springer, 2020.

[31] Marco Dorigo and Thomas Stützle. *Ant colony optimization*. MIT Press, 2004.

[32] Niklas Eén and Niklas Sörensson. "An Extensible SAT-solver". In: *SAT*. Vol. 2919. Lecture Notes in Computer Science. Springer, 2003, pp. 502–518.

[33] A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing, Second Edition*. Natural Computing Series. Springer, 2015.

[34] A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing, Second Edition*. Natural Computing Series. Springer, 2015.

[35] Matthew C. Fontaine, Ruilin Liu, Ahmed Khalifa, Jignesh Modi, Julian Togelius, Amy K. Hoover, and Stefanos Nikolaidis. "Illuminating Mario Scenes in the Latent Space of a Generative Adversarial Network". In: *AAAI*. AAAI Press, 2021, pp. 5922–5930.

[36] Matthew C. Fontaine, Julian Togelius, Stefanos Nikolaidis, and Amy K. Hoover. "Covariance matrix adaptation for the rapid illumination of behavior space". In: *GECCO*. ACM, 2020, pp. 94–102.

[37] Tobias Friedrich, Timo Kötzing, J. A. Gregor Lagodzinski, Frank Neumann, and Martin Schirneck. "Analysis of the (1 + 1) EA on subclasses of linear functions under uniform and linear constraints". In: *Theor. Comput. Sci.* 832 (2020), pp. 3–19.

[38] Tobias Friedrich, Anton Krohmer, Ralf Rothenberger, Thomas Sauerwald, and Andrew M. Sutton. "Bounds on the Satisfiability Threshold for Power Law Distributed Random SAT". In: *ESA*. Vol. 87. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 37:1–37:15.

[39] Tobias Friedrich, Anton Krohmer, Ralf Rothenberger, and Andrew M. Sutton. "Phase Transitions for Scale-Free SAT Formulas". In: *AAAI*. AAAI Press, 2017, pp. 3893–3899.

[40] Theodoros Galanos, Antonios Liapis, Georgios N. Yannakakis, and Reinhard Koenig. "ARCH-Elites: quality-diversity for urban design". In: *GECCO Companion*. ACM, 2021, pp. 313–314.

[41] Wanru Gao, Samadhi Nallaperuma, and Frank Neumann. "Feature-Based Diversity Optimization for Problem Instance Classification". In: *Evol. Comput.* 29.1 (2021), pp. 107–128.

[42] Fred W. Glover and Manuel Laguna. *Tabu Search*. Kluwer, 1997.

[43] Teofilo F. Gonzalez, ed. *Handbook of Approximation Algorithms and Metaheuristics, Second Edition, Volume 1: Methologies and Traditional Applications*. Chapman and Hall/CRC, 2018.

[44] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. "Optimal control of plotting and drilling machines: A case study". In: *ZOR Methods Model. Oper. Res.* 35.1 (1991), pp. 61–84.

[45] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual.* 2021. URL: https://www.gurobi.com.

[46] Georges R. Harik. "Finding Multimodal Solutions Using Restricted Tournament Selection". In: *ICGA*. Morgan Kaufmann, 1995, pp. 24–31.

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

[48] Keld Helsgaun. "An effective implementation of the Lin-Kernighan traveling salesman heuristic". In: *Eur. J. Oper. Res.* 126.1 (2000), pp. 106–130.

[49] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* MIT Press, 1992.

[50] Christian Horoba. "Analysis of a simple evolutionary algorithm for the multiobjective shortest path problem". In: *FOGA*. ACM, 2009, pp. 113–120.

[51] Ellis Horowitz and Sartaj Sahni. "Computing Partitions with Applications to the Knapsack Problem". In: *J. ACM* 21.2 (1974), pp. 277–292.

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

[53] James Kennedy and Russell Eberhart. "Particle swarm optimization". In: *ICNN*. IEEE, 1995, pp. 1942–1948.

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

[55] C Koulamas, SR Antony, and R Jaen. "A survey of simulated annealing applications to operations research problems". In: *Omega* 22.1 (1994), pp. 41–56.

[56] Joel Lehman and Kenneth O. Stanley. "Abandoning Objectives: Evolution Through the Search for Novelty Alone". In: *Evol. Comput.* 19.2 (2011), pp. 189–223.

[57] Xiaodong Li, Michael G. Epitropakis, Kalyanmoy Deb, and Andries P. Engelbrecht. "Seeking Multiple Solutions: An Updated Survey on Niching Methods and Their Applications". In: *IEEE Trans. Evol. Comput.* 21.4 (2017), pp. 518–538.

[58] Shen Lin and Brian W. Kernighan. "An Effective Heuristic Algorithm for the Traveling-Salesman Problem". In: *Oper. Res.* 21.2 (1973), pp. 498–516.

[59] Manuel López-Ibáñez, Thomas Stützle, and Marco Dorigo. "Ant Colony Optimization: A Component-Wise Overview". In: *Handbook of Heuristics.* Springer, 2018, pp. 371–407.

[60] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. "Iterated Local Search". In: *Handbook of Metaheuristics.* Vol. 57. International Series in Operations Research & Management Science. Kluwer / Springer, 2003, pp. 320–353.

[61] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. "The irace package: Iterated racing for automatic algorithm configuration". In: *Operations Research Perspectives* 3 (2016), pp. 43–58.

[62] Alenrex Maity and Swagatam Das. "Efficient hybrid local search heuristics for solving the travelling thief problem". In: *Appl. Soft Comput.* 93 (2020), p. 106284.

[63] Kim-Fung Man, Wallace Kit-Sang Tang, and Sam Kwong. "Genetic algorithms: concepts and applications [in engineering design]". In: *IEEE Trans. Ind. Electron.* 43.5 (1996), pp. 519–534.

[64] Zbigniew Michalewicz, Robert Hinterding, and Maciej Michalewicz. "Evolutionary algorithms". In: *Fuzzy evolutionary computation* (1997), pp. 3–31.

[65] C. E. Miller, A. W. Tucker, and R. A. Zemlin. "Integer Programming Formulation of Traveling Salesman Problems". In: *J. ACM* 7.4 (1960), pp. 326–329.

[66] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. "Chaff: Engineering an Efficient SAT Solver". In: *Proceedings of the 38th Design Automation Conference DAC.* ACM, 2001, pp. 530–535.

[67] Heinz Mühlenbein. "How Genetic Algorithms Really Work: Mutation and Hillclimbing". In: *PPSN.* Elsevier, 1992, pp. 15–26.

[68] Alexander Nadel. "Generating Diverse Solutions in SAT". In: *SAT.* Vol. 6695. Lecture Notes in Computer Science. Springer, 2011, pp. 287–301.

[69] Yuichi Nagata. "Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem". In: *Proceeding of 7th International Conference on Genetic Algorithms, 1997.* 1997.

[70] Yuichi Nagata. "High-Order Entropy-Based Population Diversity Measures in the Traveling Salesman Problem". In: *Evol. Comput.* 28.4 (2020), pp. 595–619.

[71] Yuichi Nagata. "New EAX Crossover for Large TSP Instances". In: *PPSN.* Vol. 4193. Lecture Notes in Computer Science. Springer, 2006, pp. 372–381.

[72] Yuichi Nagata and Shigenobu Kobayashi. "A Powerful Genetic Algorithm Using Edge Assembly Crossover for the Traveling Salesman Problem". In: *INFORMS J. Comput.* 25.2 (2013), pp. 346–363.

[73] Majid Namazi, M. A. Hakim Newton, Abdul Sattar, and Conrad Sanderson. "A Profit Guided Coordination Heuristic for Travelling Thief Problems". In: *SOCS.* AAAI Press, 2019, pp. 140–144.

[74] Majid Namazi, Conrad Sanderson, M. A. Hakim Newton, and Abdul Sattar. "Surrogate Assisted Optimisation for Travelling Thief Problems". In: *SOCS.* AAAI Press, 2020, pp. 111–115.

[75] Slawomir J. Nasuto and J. Mark Bishop. “Stabilizing Swarm Intelligence Search via Positive Feedback Resource Allocation”. In: *NICSO*. Vol. 129. Studies in Computational Intelligence. Springer, 2007, pp. 115–123.

[76] Aneta Neumann, Jakob Bossek, and Frank Neumann. “Diversifying greedy sampling and evolutionary diversity optimisation for constrained monotone submodular functions”. In: *GECCO*. ACM, 2021, pp. 261–269.

[77] Aneta Neumann, Wanru Gao, Carola Doerr, Frank Neumann, and Markus Wagner. “Discrepancy-based evolutionary diversity optimization”. In: *GECCO*. ACM, 2018, pp. 991–998.

[78] Aneta Neumann, Wanru Gao, Markus Wagner, and Frank Neumann. “Evolutionary diversity optimization using multi-objective indicators”. In: *GECCO*. ACM, 2019, pp. 837–845.

[79] Aneta Neumann, Sharlotte Gounder, Xiankun Yan, Gregory Sherman, Benjamin Campbell, Mingyu Guo, and Frank Neumann. “Evolutionary Diversity Optimization for the Detection and Concealment of Spatially Defined Communication Networks”. In: *GECCO*. ACM, 2023.

[80] Aneta Neumann, Zygmunt L Szpak, Wojciech Chojnacki, and Frank Neumann. “Evolutionary image composition using feature covariance matrices”. In: *GECCO*. 2017, pp. 817–824.

[81] Frank Neumann, Sergey Polyakovskiy, Martin Skutella, Leen Stougie, and Junhua Wu. “A Fully Polynomial Time Approximation Scheme for Packing While Traveling”. In: *ALGOCLOUD*. Vol. 11409. Lecture Notes in Computer Science. Springer, 2018, pp. 59–72.

[82] Adel Nikfarjam, Jakob Bossek, Aneta Neumann, and Frank Neumann. “Computing diverse sets of high quality TSP tours by EAX-based evolutionary diversity optimisation”. In: *FOGA*. ACM, 2021, 9:1–9:11.

[83] Adel Nikfarjam, Jakob Bossek, Aneta Neumann, and Frank Neumann. “Entropy-based evolutionary diversity optimisation for the traveling salesperson problem”. In: *GECCO*. ACM, 2021, pp. 600–608.

[84] Adel Nikfarjam, Anh Viet Do, and Frank Neumann. “Analysis of Quality Diversity Algorithms for the Knapsack Problem”. In: *PPSN (2)*. Vol. 13399. Lecture Notes in Computer Science. Springer, 2022, pp. 413–427.

[85] Adel Nikfarjam, Amirhossein Moosavi, Aneta Neumann, and Frank Neumann. “Computing High-Quality Solutions for the Patient Admission Scheduling Problem Using Evolutionary Diversity Optimisation”. In: *PPSN (1)*. Vol. 13398. Lecture Notes in Computer Science. Springer, 2022, pp. 250–264.

[86] Adel Nikfarjam, Aneta Neumann, Jakob Bossek, and Frank Neumann. “Co-evolutionary Diversity Optimisation for the Traveling Thief Problem”. In: *PPSN (1)*. Vol. 13398. Lecture Notes in Computer Science. Springer, 2022, pp. 237–249.

[87] Adel Nikfarjam, Aneta Neumann, and Frank Neumann. “Evolutionary diversity optimisation for the traveling thief problem”. In: *GECCO*. ACM, 2022, pp. 749–756.

[88] Adel Nikfarjam, Aneta Neumann, and Frank Neumann. “On the use of quality diversity algorithms for the traveling thief problem”. In: *GECCO*. ACM, 2022, pp. 260–268.

[89] Adel Nikfarjam, Ralf Rothenberger, Frank Neumann, and Tobias Friedrich. “Evolutionary Diversity Optimisation in Constructing Satisfying Assignments”. In: *GECCO*. ACM, 2023.

[90] I. M. Oliver, D. J. Smith, and J. R. C. Holland. “A Study of Permutation Crossover Operators on the Traveling Salesman Problem”. In: *ICGA*. Lawrence Erlbaum Associates, 1987, pp. 224–230.

[91] Ibrahim H. Osman and Gilbert Laporte. “Metaheuristics: A bibliography”. In: *Ann. Oper. Res.* 63.5 (1996), pp. 511–623.

[92] Károly F. Pál. “Selection Schemes with Spatial Isolation for Genetic Optimization”. In: *PPSN*. Vol. 866. Lecture Notes in Computer Science. Springer, 1994, pp. 170–179.

[93] Sergey Polyakovskiy, Mohammad Reza Bonyadi, Markus Wagner, Zbigniew Michalewicz, and Frank Neumann. “A comprehensive benchmark set and heuristics for the traveling thief problem”. In: *GECCO*. ACM, 2014, pp. 477–484.

[94] Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. “Quality Diversity: A New Frontier for Evolutionary Computation”. In: *Frontiers Robotics AI* 3 (2016), p. 40.

[95] Justin K. Pugh, Lisa B. Soros, Paul A. Szerlip, and Kenneth O. Stanley. “Confronting the Challenge of Quality Diversity”. In: *GECCO*. ACM, 2015, pp. 967–974.

[96] Nemanja Rakicevic, Antoine Cully, and Petar Kormushev. “Policy manifold search: exploring the manifold hypothesis for diversity-based neuroevolution”. In: *GECCO*. ACM, 2021, pp. 901–909.

[97] Gerhard Reinelt. “TSPLIB–A Traveling Salesman Problem Library”. In: *ORSA Journal on Computing* 3.4 (1991), pp. 376–384.

[98] Alice Richardson. *Nonparametric statistics for non-statisticians: A step-by-step approach by Gregory W. Corder, dale I. foreman.* 2010.

[99] Louis B. Rosenberg. “Human Swarms, a real-time method for collective intelligence”. In: *ECAL*. MIT Press, 2015, pp. 658–659.

[100] Keith W. Ross and Danny H. K. Tsang. “The stochastic knapsack problem”. In: *IEEE Trans. Commun.* 37.7 (1989), pp. 740–747.

[101] Franz Rothlauf. *Representations for genetic and evolutionary algorithms*. Vol. 104. Studies in Fuzziness and Soft Computing. Springer, 2002.

[102] Hussain Aziz Saleh and Rachid Chelouah. “The design of the global navigation satellite system surveying networks using genetic algorithms”. In: *Eng. Appl. Artif. Intell.* 17.1 (2004), pp. 111–122.

[103] Bruno Sareni and Laurent Krähenbühl. “Fitness sharing and niching methods revisited”. In: *IEEE Trans. Evol. Comput.* 2.3 (1998), pp. 97–106.

[104] Ruhul Sarker, Masoud Mohammadian, Xin Yao, and Ingo Wegener. *Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions.* Springer, 2002.

[105] J. David Schaffer. “Multiple Objective Optimization with Vector Evaluated Genetic Algorithms”. In: *ICGA*. Lawrence Erlbaum Associates, 1985, pp. 93–100.

[106] João P. Marques Silva and Karem A. Sakallah. “GRASP: A Search Algorithm for Propositional Satisfiability”. In: *IEEE Trans. Computers* 48.5 (1999), pp. 506–521.

[107] Gulshan Singh and Kalyanmoy Deb. “Comparison of multi-modal optimization algorithms based on evolutionary algorithms”. In: *GECCO*. ACM, 2006, pp. 1305–1312.

[108] Kirby Steckel and Jacob Schrum. “Illuminating the space of beatable lode runner levels produced by various generative adversarial networks”. In: *GECCO Companion*. ACM, 2021, pp. 111–112.

[109] Gilbert Syswerda. “Uniform Crossover in Genetic Algorithms”. In: *ICGA*. Morgan Kaufmann, 1989, pp. 2–9.

[110] Madeleine Theile. “Exact Solutions to the Traveling Salesperson Problem by a Population-Based Evolutionary Algorithm”. In: *EvoCOP*. Vol. 5482. Lecture Notes in Computer Science. Springer, 2009, pp. 145–155.

[111] Paolo Toth. “Dynamic programming algorithms for the Zero-One Knapsack Problem”. In: *Computing* 25.1 (1980), pp. 29–45.

[112] Tamara Ulrich and Lothar Thiele. “Maximizing population diversity in single-objective optimization”. In: *GECCO*. ACM, 2011, pp. 641–648.

[113] Vassilis Vassiliades, Konstantinos I. Chatzilygeroudis, and Jean-Baptiste Mouret. “Using Centroidal Voronoi Tessellations to Scale Up the Multidimensional Archive of Phenotypic Elites Algorithm”. In: *IEEE Trans. Evol. Comput.* 22.4 (2018), pp. 623–630.

[114] Vijay V. Vazirani. *Approximation algorithms.* Springer, 2001. ISBN: 978-3-540-65367-7.

[115] Markus Wagner. “Stealing Items More Efficiently with Ants: A Swarm Intelligence Approach to the Travelling Thief Problem”. In: *ANTS Conference*. Vol. 9882. Lecture Notes in Computer Science. Springer, 2016, pp. 273–281.

[116] Markus Wagner, Marius Lindauer, Mustafa Misir, Samadhi Nallaperuma, and Frank Hutter. “A case study of algorithm selection for the traveling thief problem”. In: *J. Heuristics* 24.3 (2018), pp. 295–320.

[117] Handing Wang, Yaochu Jin, and Xin Yao. “Diversity Assessment in Many-Objective Optimization”. In: *IEEE Trans. Cybern.* 47.6 (2017), pp. 1510–1522.

[118] L. Darrell Whitley, Timothy Starkweather, and D'Ann Fuquay. "Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator". In: *ICGA*. Morgan Kaufmann, 1989, pp. 133–140.

[119] Junhua Wu, Markus Wagner, Sergey Polyakovskiy, and Frank Neumann. "Exact Approaches for the Travelling Thief Problem". In: *SEAL*. Vol. 10593. Lecture Notes in Computer Science. Springer, 2017, pp. 110–121.

[120] Rogier Hans Wuijts and Dirk Thierens. "Investigation of the traveling thief problem". In: *GECCO*. ACM, 2019, pp. 329–337.

[121] Xiao-Feng Xie and Jiming Liu. "Multiagent Optimization System for Solving the Traveling Salesman Problem (TSP)". In: *IEEE Trans. Syst. Man Cybern. Part B* 39.2 (2009), pp. 489–502.

[122] Yue Xie, Aneta Neumann, Frank Neumann, and Andrew M. Sutton. "Runtime analysis of RLS and the (1+1) EA for the chance-constrained knapsack problem with correlated uniform weights". In: *GECCO*. ACM, 2021, pp. 1187–1194.

[123] Mohamed El Yafrani and Belaïd Ahiod. "Cosolver2B: An efficient local search heuristic for the Travelling Thief Problem". In: *AICCSA*. IEEE Computer Society, 2015, pp. 1–5.

[124] Mohamed El Yafrani and Belaïd Ahiod. "Efficiently solving the Traveling Thief Problem using hill climbing and simulated annealing". In: *Inf. Sci.* 432 (2018), pp. 231–244.

[125] Daniela Zaharie. "Density Based Clustering with Crowding Differential Evolution". In: *SYNASC*. IEEE Computer Society, 2005, pp. 343–350.

[126] Enrico Zardini, Davide Zappetti, Davide Zambrano, Giovanni Iacca, and Dario Floreano. "Seeking quality diversity in evolutionary co-design of morphology and control of soft tensegrity modular robots". In: *GECCO*. ACM, 2021, pp. 189–197.

[127] Kenny Qili Zhu and Ziwei Liu. "Population Diversity in Permutation-Based Genetic Algorithm". In: *ECML*. Vol. 3201. Springer, 2004, pp. 537–547.

[128] Wiem Zouari, Inès Alaya, and Moncef Tagina. "A new hybrid ant colony algorithms for the traveling thief problem". In: *GECCO (Companion)*. ACM, 2019, pp. 95–96.