1 2 9 0

UNIVERSIDADE Ð
COIMBRA

Guilherme Cardoso Gomes da Silva

OPTIMIZATION BY LEARNING

Dissertation in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems advised by Prof. Nuno Lourenço and Prof. Francisco Baptista Pereira and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering.

October 2020

This page is intentionally left blank.

Faculty of Sciences and Technology

Department of Informatics Engineering

Optimization By Learning

Guilherme Cardoso Gomes da Silva

gcsilva@student.dei.uc.pt

Dissertation in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems advised by Prof. Nuno Lourenço and Prof. Francisco Pereira presented to the
Faculty of Sciences and Technology / Department of Informatics Engineering.

October 2020

1 2 9 0

UNIVERSIDADE D COIMBRA

This page is intentionally left blank.

Acknowledgements

I would like to first and foremost thank my advisors, Professor Nuno Lourenço and Professor Francisco Pereira for their unrelenting support and constant guidance over the course of the year. Their availability and guidance whenever I needed them were what made this dissertation possible in the first place, and I can not thank them enough for the growth I have experienced during this year, both professionally, and as a person.

Thank you to Artur, Diogo, Leonardo, Pedro and Tiago for their constant presence and reassurance, for the infinite hours fun, for the incredible support, and for all the time we spend, and hopefully will keep spending, together.

A huge thank you to the Lokos, António, André, Diogo, Henrique, Vaz, Jorge, JP, JA and Bernardo, for being my friends for most of my life. For always being there for me, both in good, and bad times, and being a second family I know I can always count on if I want to have fun, or if I need a shoulder to rely on. I know that even when we are spread out across the world we will still be just as united during the course of our lives.

A special thank you to Carolina for being my very best friend, and for proving to be the most chaotic, supportive human being one could wish for. I would never have made it through this without you.

Finally, but not least important, I want to thank my family, in particular my mother and sister, for being the two strongest human beings to ever exist, and for giving me the world whenever I needed them the most.

This year brought exceptional circumstances and hardships with it, which is why, from the bottom of my heart, I really want to thank all the people mentioned here, for helping me see through this journey until the end. Thank you.

This page is intentionally left blank.

Abstract

Combinatorial optimization (CO) is a field that has been an object of study for many years, and there are a multitude of different algorithms specialized in solving a plethora of problems in the area. A typical CO problem consists of finding an optimal solution among a set of candidates that maximizes/minimizes the given objective.

The field of Machine Learning has experienced a large growth in recent years for a multitude of purposes, but of particular interest here is its usefulness in the automation of the process of creating, adapting, and selecting optimization algorithms.

In this dissertation, we explore the usage of fitness landscape analysis metrics in a local, stochastic, iterative exploration of the Travelling Salesman Problem, to train a Machine Learning classifier that attempts to select the best possible operator given the proprieties of the local landscape of a given solution. We create a dataset to train our model on, then process to analyse the generated instances and the behavior of the extracted metrics. We then test several different classifier configurations, explore the obtained results, and finally test our solution against an hill climbing algorithm.

The results show the choice of promising operators utilizing a Machine Learning model trained using fitness landscape analysis metrics is not a trivial task, due to difficulty in discerning patterns both on operator behavior and on the landscape proprieties of the studied TSP instances.

Keywords

Machine Learning, Classification, Combinatorial Optimization, Travelling Salesman Problem, Operator Selection, Evolutionary Computing, Fitness Landscape Analysis

This page is intentionally left blank.

Resumo

Optimização combinatória é uma área que é objecto de estudo há vários anos, onde existem inúmeros algoritmos diferentes que se especializam em resolver problemas dentro do domínio desta. Um problema de optimização combinatória típico consiste em encontrar uma solução óptima entre um grupo de soluções candidatas com o intuito de maximizar/minimizar um dado objectivo.

O campo de Machine Learning observou recentemente um grande crescimento, de particular relevância para este trabalho pela sua utilidade na automatização do processo de criação, adaptação e seleção de algorítmos de optimização aplicados a optimização combinatória.

Nesta tese, é explorada a proposta de utilização de métricas de análise de fitness landscape, em exploração local, estocástica e iterativa do Travelling Salesman Problem. Treinamos um classificador de Machine Learning que tenta selecionar o melhor operador possivel numa dada instância, observando as propriedades da landscape do local onde se encontra. Primeiro criamos um dataset para treinar o nosso classificador, e analisamos as instâncias generadas e os padrões de comportamento das métricas retiradas. Depois testamos diferentes configurações de classificadores, explorando os resultados obtidos, e finalmente testamos a nossa solução em conjunto com um algoritmo trepa colinas.

Os resultados mostram que a escolha de operadores promissores utilizando um modelo de Machine Learning treinado com métricas relativas à análise da fitness landscape não é uma tarefa trivial, devido a dificuldades em distinguir padrões no comportamento dos operadores e nas propriedades das landscapes dos problemas de TSP abordados aqui.

Palavras-Chave

Machine Learning, Classificação, Optimização Combinatória, Travelling Salesman Problem, Seleção de Operadores, Computação Evolucionária, Análise de Fitness Landscape

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

CO	Combinatorial Optimization. 2, 13, 51, 52
EA	Evolutionary Algorithms. 6
FDC	Fitness Distance Correlation. xiv, 17, 25, 31, 32, 39
ML	Machine Learning. 1, 2, 7, 8, 11, 13, 25, 48, 51, 52
TSP	Travelling Salesman Problem. 1–3, 7, 13, 52

This page is intentionally left blank.

List of Figures

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

Optimization is an area that has been the object of study of many scientific fields, from computer science to economics, in which the objective is to select the best solution among a pool of objects. At its core, an optimisation problem involves finding a solution that maximizes (or minimizes, depending on the problem) its objective function. A particular subfield of problems that we are interested in engaging with, is combinatorial optimization (CO). Combinatorial optimization problems seek to find the optimum object in a finite set of possible solutions. The number of objects in said problems is too big to effectively be searched extensively, and as such, more creative solutions are needed to solve them [36].

One example of such problems, and perhaps the most famous, is the Travelling Salesman Problem (TSP), which, given a list of cities and the distance between each pair, asks what is the shortest possible route that visits all cities once, returning to the city where it first started. This problem has garnered interest in the field of computer science since its introduction [6], due to several factors, such as its applicability in real-world problems ([9], [20]).

In this work, we want to focus on local, iterative, and stochastic optimization algorithms. In particular, iterative stochastic optimization. As the name implies, such methods incorporate a random component, be it in its initialization, on the search steps, or in both [17]. The application of operators to modify a given solution in stochastic methods typically involves random elements, and there is often a need to define what are the best operators to apply at a given step.

With the ever growing amount of data available, one area that has garnered interest over recent years is that of Machine Learning (ML), which nowadays is incorporated in numerous fields, including in aiding the process of designing optimization algorithms [10]. There is a high number of decisions that have to be be hand-made, often empirically, in order to tackle any particular problem. Introducing automated ways to ease some of these decisions can, as such, be a valuable contribution to the field.

1.1 Objectives

The objective of this dissertation is to automatize the process of optimizing the selection of operators in a given CO algorithm, incorporating ML elements in its solution, utilizing fitness landscape analysis metrics. The focus will be the TSP, but one of the objectives is to implement a framework that is broad enough to generalize to other optimization problems in the future. We will focus our attention on the usage of supervised learning models, and as such, we start by creating a dataset of solutions for three distinct TSP instances using different heuristics for the initial generation of the solutions. Each solution is then modified by the application of perturbation operators. After a certain number of steps we record a series of metrics concerned with the fitness landscape properties, and use the generated examples to create a dataset which we train our ML model on.

Afterwards we perform an extensive analysis of the generated datasets, including the metrics used during their creation process, to extract relevant insight and evaluate the features for their potential relevance, which we will then use to develop our ML solution. We then attempt to develop a classifier that incorporates ML to tackle the TSP, by making an informed decision on which operator to choose based on the gathered metrics. Finally, we perform experiences with the resulting classifier, and assess its viability compared to other standard local search algorithms. The resulting model shows low capability in distinguishing optimal operators to be used, due to difficulty in discerning patterns on operator behavior.

1.2 Main Contributions

The main contributions of our work are as follows:

1. Proposal of a ML framework that utilizes metrics related to the properties of fitness landscape, in order to aid local optimization algorithms in the prediction of the most promising operator to be used at any step during its search.
2. Creation of a dataset from various TSP problems, with several generated instances that gather local fitness landscape information of the problem, by performing a small random walk across a given problem instance.
3. Exploration of the gathered data for behavioral patterns of the explored metrics and their relations with one another, and the analysis of the distribution of operator usage across the different metrics.
4. Creation of a Random Forest ML model that utilizes Random Forests, and is trained on the created data to evaluate what operator to use, and comparison of created model with a standard hill climbing algorithm.

1.3 Document Structure

From this point on, the document is structured as follows: Chapter 2 aims to provide context on the different topics approached in this dissertation. In particular, Section 2.1 provides background information on Combinatorial Optimization (CO), and particularly, the TSP. Section 2.2 describes a number of Machine Learning concepts necessary for later chapters. Section 2.3 introduces current research on the topic of solving the TSP resorting to ML solutions. Chapter 3 is used to provide information on the approach we used to tackle the problem, how the dataset was built, what metrics were utilized and how the tour generation was done. Chapter 4 encapsulates the analysis done on the generated dataset, the exploration of ML solutions, and the comparison of our best solution to a standard hill climbing algorithm. Finally, Section 5 is reserved for concluding remarks on the dissertation, the discussion of results, and possible paths for the expansion of this work.

Chapter 2

Background

This chapter presents key concepts necessary to provide context and background to the remainder of the thesis. As such, section 2.1 is dedicated to introducing combinatorial optimization and the Traveling Salesman Problem in detail, including several different algorithms and approaches to solve it. Section 2.2 explores the area of Machine Learning and introduces concepts that are relevant to the implementation details of the thesis. Finally, Section 2.3 details previous research that was deemed valuable and worthy of mention to our work.

2.1 Combinatorial Optimization

Combinatorial Optimization (CO) is historically an area that impacts a large variety of scientific fields, from bioinformatics to electronic commerce ([16]). The problems presented in this area have the objective of finding a solution amongst a finite and discrete set of candidates in order to maximize or minimize a given objective function. This function is defined on candidate solutions, and its value is the measure of the solution's quality [16].

Finding a solution among a set of candidates is not a trivial task, since the space of possible solutions is usually too large. This particular characteristic makes the use of brute force to find the optimal solution unfeasible, which makes the use of heuristic algorithms preferable.

Examples of famous CO problems include the Knapsack problem, for example, which given a set of items with a weight w and value c assigned to them, and a knapsack of maximum capacity C, seeks to find the subset of items that fit in the knapsack with the maximum possible value [23]; and the Travelling Salesman Problem (TSP), which is described in detail on the next subsection.

2.1.1 Traveling Salesman Problem

The TSP is a NP-hard problem, meaning it is at least as hard as the hardest NP problems. NP (non-deterministic polynomial) is the set of decision problems for which their solution is verifiable using a deterministic algorithm in polynomial time [3].

Starting from an initial city, the problem consists of finding the optimal route (or, in other words, the shortest tour) by visiting a set of cities only once. As a more formal description, the TSP can be described as a graph G = *(C, E)* where C are the vertices, representing the cities the salesman has to travel through, and E is the representation of the edges that connect each city pair, with their weight representing the distance between the two cities in said pair. The problem can then be described as finding an Hamiltonian cycle with the least weight possible, by visiting each node $c \in C$ only once.

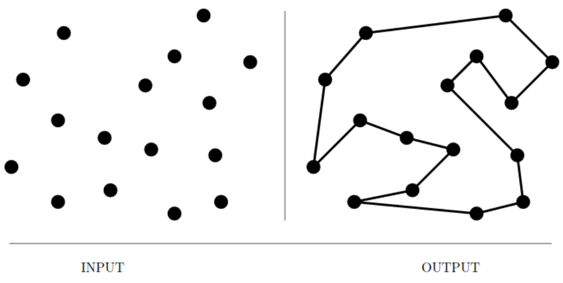


Figure 2.1: An example of a TSP solution [38].

There are many variations of the TSP but we will focus exclusively on symmetric TSP instances, where the weight of each city pair is the same in both directions.

The TSP has a large number of real world applications, ranging from X-Ray Crystallography [9] to the design of cell manufacturing systems [20], which serves as a testament to the flexibility and applicability of the TSP. This, combined with its relatively simple premise and implementation difficulty, makes the problem an adequate starting point for the development of the work found in this thesis.

While a relatively simple problem to state, a TSP with n cities has a set of candidate solutions of $(n$ - $1)!/2$ tours, which is a number that grows exponentially as the number of cities increases. As such, as is often the case with CO problems, solutions to find the optimal tours other than exhaustive search were developed out of necessity. We will mention some of the methods that are historically relevant in the following section, dividing them into two distinct categories: exact algorithms and heuristic algorithms.

Exact Algorithms

Exact algorithms, as the name implies, guarantee an optimal solution as their output. Historically, the exact algorithms that had the most success in solving the TSP are cutting-plane and facet finding algorithms [5]. The state-of-the-art solver Concorde [5], in fact, applies cutting-plane methods iteratively to solve linear programming relaxations of the problem, combined with a branch-and-bound approach, until it reaches the optimal solution.

These algorithms can be extremely complex, and take a long time to solve large instances of the TSP, in addition to requiring large amounts of computational power, and as such are often impractical. As such, heuristic algorithms were developed as alternatives that seek an approximate solution using faster and more efficient methods.

Heuristic Algorithms

Heuristic algorithms, or approximation algorithms, do not guarantee an optimal solution, but instead offer solutions that seek to approximate its quality to the optimal one.

On the TSP specifically, depending on their behavior, such algorithms can be further divided in different subcategories as well, such as constructive heuristics, relating to the building of the tours, and iterative heuristics, related to the improvement of solutions over the algorithms' runtime.

For initial solution construction, commonly used constructors are *random*, which randomly selects edges with no regard for their quality, as long as it considers the tour a valid one; *Nearest neighbour search*, which starting at a random city, searches for a non-visited city and picks the one closest to the current one, adding it to the tour, finally closing the tour connecting the first city to the last chosen one [15]; Or *multiple fragment*, which, similarly to the nearest neighbour heuristic, considers edges by increasing order of length, but unlike the former, maintains a set of fragmented tours [8], hence its name.

Lin-Kernighan Heuristic

Notable examples of tour improvement heuristics are the k-opt methods, which exchange k-edges on a TSP tour iteratively to create a new, shorter tour, until no more operations of the sort can be applied to the solution. Figure 2.2 displays a basic example of a 2-opt swap, where two edges are considered, and if swapping them results in a lower tour cost, then the operation is performed. Such edge flipping methods are the basis of the *Lin-Kernighan heuristic*, which is an heuristic that proved to be successful in solving a large number of TSP instances with good results. The idea of the LK algorithm is to consider a growing number of edge exchanges to be performed at each step, in such a way that a feasible tour can be achieved at any step during the process [15]. By considering two sets of edges X and Y, the objective is to replace edges X by the set of edges Y in a way that reduces the overall cost of the tour [25]. The changes are performed iteratively until no further improvements can be found, following a number of criteria that guarantee the feasibility of the solution along every step of the way. This method is considered a *local optimization algorithm*, a concept that will be expanded on further below.

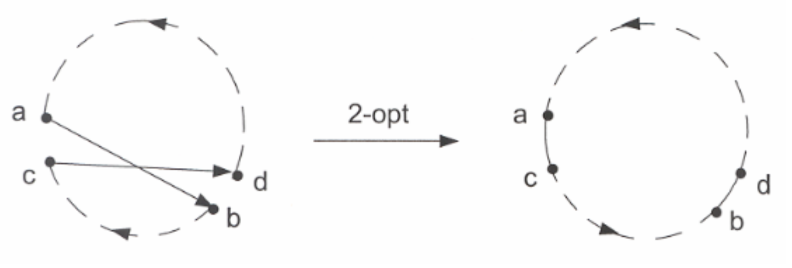


Figure 2.2: An example of a simple 2-opt move found in [44].

Over the years, several refinements were introduced to the algorithm to improve its results for TSP instances with larger numbers of cities. For example, the introduction of a measurement of *nearness* utilizing minimum spanning trees on [15], or the introduction of *"kicks"* in a tour to act as a perturbation, as well as adjustments to the algorithm's backtracking capabilities and different heuristics used to generate the initial solution tours on [4], all contributed to allow the algorithm to run in acceptable time and obtain good results on large TSP instances.

Cristofides Heuristic

Another heuristic algorithm of great importance is the *Christofides–Serdyukov* algorithm. It is an algorithm proven to find approximate solutions to symmetric TSP instances on a metric space within a range of 50% of the optimal solution, provided it satisfies the triangle inequality [11]. It is considered one of the best general heuristics for the TSP since its formulation, with only very recent research improving on it, even if by an extremely small amount [21]. The way the algorithm is designed is described in Algorithm 1.

Algorithm 1: Christofides heuristic for tour generation. [14]

```
Construct minimum spanning tree, $M$, for a graph $G$;
Generate sub-graph $H$ from the vertices from $G$ with an odd degree of vertices;
Create minimum-cost perfect matching $P$ in $H$;
Unite $P$ and $M$ to form Eulerian multigraph;
Calculate Euler tour;
Remove repeated vertices
```

Evolutionary Algorithms

Evolutionary Algorithms (EA) are, as the name suggests, loosely inspired on the theory of evolution observed in the natural world [13]. EAs resort to a population of individuals within a given search space which evolve across generations, according to a quality function to be maximized. The population is randomly generated in the beginning and evolves using the top individuals as seeds for the next generation through mutations and crossovers. Crossovers are applied to two (or more) individuals and combine information from both parents to generate offspring. Mutation is applied in one individual and results in a new individual. The generic algorithm can be represented as seen in Algorithm 2.

Algorithm 2: Generic Evolutionary Algorithm

```
initialize population;
evaluate quality;
while termination condition not met do
    select parents;
    crossover pairs of individuals;
    mutate offspring;
    evaluate new individuals;
    replace individuals for next generation;
end
return best individuals;
```

While standard EAs started as defined in Algorithm 2, many variants have since appeared, such as **Genetic Programming** (GP), that uses trees as chromosomes for its representation, and can be used in syntax of arithmetic expressions, formulas, or even code written in a programming language [13], among others.

EAs are described in much greater detail in [13].

Evolutionary algorithms have historically been used as an attempt to solve the TSP ([40], [47]), and prove to be reliable in finding an approximate solution in an efficient time frame, with the most promising results being the works of [32], which will be further expanded upon below.

Local Search

Local search algorithms, as the name implies, focus on applying local changes to solutions in the search space until no more changes are found (reached a local maximum), or the maximum number of iterations has been reached. An example of a local search algorithm is the previously mentioned 2-opt algorithm that works on specific edges of a given solution.

Local search methods have been used on the TSP as a testing ground for many decades ([34], [27]). Hill climbing algorithms are a local search method that start with an arbitrary solution, and then attempt to apply incremental changes to it. A fitness function is used to determine if the quality of the solution improved or declined, and if it presents higher quality than the previous solution, the new solution is chosen and the process repeats until no new improvements are found, or the maximum allotted time for the algorithm to run reaches its end [35].

Hill climbers have a simple implementation and are widely used in a variety of problems, but are not without their issues. It is very easy for a hill climber to fall into local optima, and as such they often present sub optimal results in problems with an irregular search space, meaning there are a lot of local optima and plateaus where a solution can get stuck, which is often the case in problems such as the TSP.

Variations to this algorithm that seek to attenuate this problem include simulated annealing, for example, which involves accepting seemingly worse solutions while first searching the solution space, allowing for a more extensive exploration, and decreasing the probability of accepting worse solutions as time goes on, in order to converge on better solutions while escaping local traps regular hill climbers tend to fall into.

The modification and improvement of such algorithms with perturbations to the solutions and "kicks" to avoid local maxima found in the search space are explored in *Iterated Local Search (ILS)*, a meta-heuristic that explores different ways to build upon local search algorithms to improve their performance [26].

2.2 Machine Learning

Machine Learning (ML) is an area of Computer Science that attempts to optimize a computer's performance by having it learn from past examples and already available data. It is used to build mathematical models, using statistical data, to either predict future events or further explore available information [35]. The area rose in prominence in recent years, and has found success in a wide variety of fields, such as veichle automation (see [28] as an example).

The objective of this section is to give a brief description of different Machine Learning concepts, in order to provide context on topics that are explored in this dissertation.

Supervised Learning

Supervised learning is a Machine Learning (ML) area that accepts a set of data instances that are correctly labeled, referred to as training data, to learn relations and behaviors of the features that describe the given instances. The task is then to infer a function from the training data and apply said function to correctly predict the values of the testing data, composed of unlabeled instances, and categorizing them correctly. Supervised learning is the most commonly used type of learning for classification and regression [30], and is the most appropriate for the work presented here.

Statistical Classification

Classification problems are related to the task of identification, and categorization of unlabeled instances, sorting them into appropriate classes, according to a set of rules constructed on previously observed patterns and behaviors found in already explored, and properly labeled, data [2].

ML algorithms that deal with the task of classification are known as classifiers. Of particular importance to us are decision tree classifiers, which we will describe below in greater detail.

Decision Trees

A decision tree is a supervised, predictive model that attempts to predict the correct class of unlabeled instances of a given item, observing its features and attributing a label according to previously constructed decision rules based on labeled data. A tree is built top-down, starting with its root node. At every node the features of the dataset are evaluated and a split condition is decided based on a gain measure that quantifies the improvement on said split [37]. The gain measure we use in this work is the Gini Impurity, which we briefly describe below. Such process is done iteratively until the tree is complete, which is when the leaf nodes cannot be split further, or doing so does not add significant value to the labeling of examples, meaning a criterion cannot be found on which to further increase the tree's labeling capacity, thus an example that falls on said leaf is classified with the majority class present in that node. A simple example of a decision tree is found on Fig. 2.3.

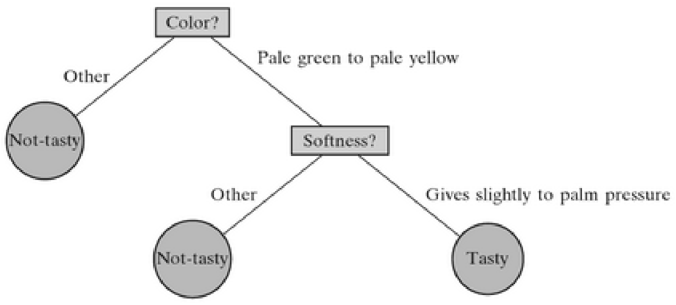


Figure 2.3: An example of a decision tree built to determine if a papaya is tasty or not from [37]. Note the top-down order of the analysed features, from most to least relevant in identifying the correct label.

Decision trees can present some disadvantages, such as a tendency to overfit on the training set, and might display lower predictive accuracy than some other classification methods [18]. However, methods such as random forest creation can attenuate such issues, and their interpretability, and overall good performance with large datasets [18], make them a powerful tool that fits the framework of this dissertation.

Gini Impurity

As previously mentioned, a decision tree bases the splitting of its internal nodes on a specific criteiron called the gain value. One of the used criteria on this dissertation is the **Gini Impurity** measure. Equation 2.1 describes how to calculate the Gini Impurity, which measures the probability of an element in the dataset being incorrectly labeled according to the class distribution in the dataset, assuming C is the total number of classes, and *p(i)* is the probability of labeling an instance with class i.

$$G = \sum_{i=1}^{C} p(i) * (1 - p(i)) \tag{2.1}$$

A decision tree bases its splits on maximizing information gain. At every split the weighted Gini Impurity of each side of the split is calculated, and the results are then subtracted to the original impurity. The higher the gain, the better the split is considered, and the node's split rule is decided based on the best possible split.

Random Forests

Random forests are an ensemble method that reduce the danger of overfitting by creating an ensemble of trees, and then allowing each tree to classify unlabeled instances of a problem with only a subset of the features of the instances to inform their decisions on split nodes. This, coupled with sampling subsets of the data to train the different trees on, allows the model to predict unlabeled data by aggregating the results of different trees in the forest. Typically, m features are chosen from the whole set of feature p as $m \approx \sqrt{p}$ [18]. Random forests are an improvement over single decision tree predictions because of their capability to homogenize the different trees' predictions and reduce variance on the results, making for more robust predictors overall and substantially attenuating problems found on decision tree classifiers like overfitting [18]. As such, we will implement random forests in our work as a way to smoothen the obtained results.

2.3 Related Work

This section is dedicated to tying recent research with our work, showing the advancements done in recent years when it comes to combining machine learning and combinatorial optimization. As such, in this chapter we discuss papers that are relevant to our dissertation in the area of fitness landscape analysis, machine learning techniques, algorithm selection, and evolutionary algorithms.

When it comes to current research in evolutionary algorithms solving the TSP, works such as [32] are important to highlight. In it, the authors introduce a genetic algorithm (GA) that utilizes an edge assembly crossover (EAX), originally developed in [31], and enhance it further by introducing additional improvements such as local-search methods to the EAX operator to increase its efficiency and running time, and maintenance of population diversity via adaptation of the notion of entropy on a given population. The resulting GA is capable of finding optimal or best-known solutions for TSP benchmark instances with up to 200 000 cities. Notably, it does not use the LK heuristic in its implementation, which marks a remarkable achievement in the exploration of heuristic approaches to the TSP.

Another important contribution to our work is the notion of fitness landscape. A term originally coined by Wright ([46]), fitness landscape analysis attempts to describe the search space as a landscape to be traversed by an algorithm. This spanned a number of different works in numerous areas that seek to use said metrics to further improve algorithm performance.

One particular work we want to highlight is [41]. This paper utilizes the concept of fitness landscape to help predict the performance of an evolutionary algorithm on a Multidimensional Knapsack problem. The focus of the work is then to study the behavior of the fitness landscape using different representations for a set of operators typically used in problems of such nature, and evaluate the results of said landscapes.

While its focus is the discovery of optimal combinations of representation, operators and heuristics using landscape analysis, of particular relevance to our dissertation is how the combination of several landscape features such as distance to optimum, fitness distance correlation and autocorrelation of solutions is used to aid in the exploration of results. We will make use of these features, describing them in greater detail in Chapter 3.

When it comes to the exploration of the TSP landscape, works such as [39], serve as early examples of capturing the properties of different TSP landscapes. More recently, the works of [42] examine a large number of properties such as the local search operator, the number of local optima, and the distance between optima, on different instances of the problem in order to aid in the choice of proper algorithm selection based on a given problem.

On the topic of Machine Learning, we can highlight works such as Bello *et al.*'s article [7], which combines a Reinforcement Learning (RL) approach with a Neural Network, to create a framework to tackle combinatorial optimization problems such as the TSP, based on policy gradient approaches, and achieves close to optimal results on up to 100 nodes. The authors consider two approaches: the first uses a training set to optimize a Recurrent Neural Network (RNN), which uses the expected reward as the objective. Based on [43] Pointer Networks, it encodes each city as an input. At each step it then produces a distribution of what city to visit next in the tour, using a pointing mechanism, which is then again passed as an input for the next step. They then use such an architecture to learn the best stochastic policy parameters that, given a set of points, assign high probabilities to short tours and low probabilities to long tours. The second approach, which the authors call *active search* uses model-free policy based RL to optimize the parameters of the Pointer Network, and then resort to policy gradient methods and stochastic gradient descent to fine tune parameters. Starting from a random policy, it iteratively optimizes the RNN (using the expected reward objective again), keeping track of the best solution sampled at the same time.

Another RL based approach is that of Deudon *et al.* [12], which investigates a framework to learn heuristics for combinatorial tasks, via training a Neural Network using reinforcement learning policies, further enhancing the heuristics learned by the model with 2-opt local search to improve running time. The model is tested on TSP maps with 20, 50 and 100 cities. The results are close to the state of the art Concorde solution and present a reasonable speedup compared to Bello *et al.*'s work mentioned previously, which reinforces the notion that Machine Learning tools are powerful when capturing problem structure on CO problems, being capable of producing good quality solutions in a reasonable amount of runtime.

The combination of the study of algorithmic selection coupled with Machine Learning is also an important research topic. The works of Kotthoff *et al.* [24] and their follow up [22] focus on introducing algorithm selection techniques on a per-instance basis of the TSP, resorting to machine learning techniques to select exactly one algorithm per TSP instance. The work utilizes an elevated number of features (114 features on [24] and 405 on [22]), ranging from features related to the spatial distribution of nodes on an Euclidean plane, to features introduced in recent years such as edge lengths of convex hulls ([33]), and then uses three supervised learning strategies, classification, regression and paired regression, to select the appropriate algorithms for a given TSP instance. The results show that the higher performing selectors are two Support Vector Machines (SVM), that highlight the features from [33] as the most relevant features for a basis to the selection of algorithms.

These works highlight the state of current research in the field of TSP optimization. As we can see, the combination of machine learning tools and algorithm selection have seen plenty of use, and there is a history of analysis of the fitness landscape of the TSP. However, there seems to exist an opportunity in the current literature to explore the possibility of making use of ML methods combined with the fitness landscape metrics we are utilizing, to create classifiers that choose an operator to be applied to an instance at a given step based on previously observed patterns, in a reasonable runtime.

This page is intentionally left blank.

Chapter 3

Approach

The main goal of this work is to gauge the value of combining features related to the fitness landscape analysis and ML for the task of deciding optimal operator usage, using information retrieved by locally exploring the landscape of a given Combinatorial Optimization (CO) problem. Figure 3.1 shows an overview of the steps taken to gain a deeper understanding of how to reach our objective.

The problem we set out to solve is the automation of operator selection. In order to do this, there is a need to choose a problem in order to explore and retrieve metrics relevant to our work. To this end, we use the TSP as the testing ground for our framework.

In order to study the behavior of the fitness landscape metrics, three different instances of TSP problems were chosen. We create a set of initial tours with different heuristics, and perform a random walk with each in order to explore the properties of the local landscape of the TSP instance, by applying three different mutation operators successively on every generated solution. This allows us to gather information about the fitness landscape properties of the different maps. The goal is then to tie the patterns observed in the extracted metrics with the chosen mutation operators, and use such information alongside a ML solution, and have the resulting model be capable of making informed decisions on what operator to use at a given step based on the local properties of the landscape where the solution being evaluated is found.

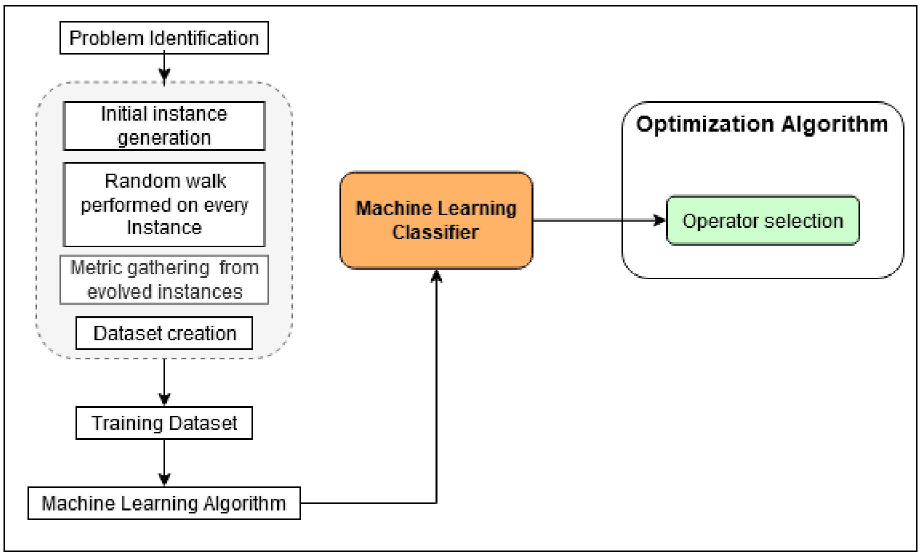

Problem Identification
Initial instance generation
Random walk performed on every Instance
Metric gathering from evolved instances
Dataset creation
Training Dataset
Machine Learning Algorithm
Machine Learning Classifier
Optimization Algorithm
Operator selection

Figure 3.1: Overview of the approach used in this work.

This chapter is thus dedicated to the detailing of the dataset creation and example generation. Section 3.1 details the used TSP instances, 3.2 describes the different methods used for tour generation, Section 3.3 explores the selected operators to be analysed. Finally, Section 3.4 goes into detail on what metrics were extracted from the generated examples and how they are presented in the dataset.

3.1 TSP Instances

The basis for our example generation were three TSP maps from the **TSPLIB** library [1], a library consisting of various instances of Symmetric Travelling Salesman Problems. Each instance includes a number of cities with their respective coordinates on each file. We selected three instances of the problem, *eil51*, *eil76* and *ts225*, each with 51, 76. and 225 cities respectively. These instances were chosen based on several different factors - Firstly, testing on different instances of the TSP was considered a priority, so we would able to explore the obtained metrics on landscapes with different properties to maximize information gain, which is why three distinct sets were chosen; Secondly, the presented instances include the optimal found solution, which is necessary for some of the metrics we want to analyze (see section 3.4). If the optimal solution is not available or known, an estimate of the optimal solution using known heuristics should suffice as a starting point, but for simplicity we chose maps with already available optimal tours in this dissertation; Thirdly, the solutions are of an acceptable length to allow a high number of possible tours, while still being of small enough length to be explored in a reasonable amount of time, even with computational limitations. We choose permutation representation for the tours, as it facilitates the application of the selected operators described in Section 3.3; Finally, the *ts225* problem instance is atypical as it features a large number of local optima [29], and is suitable to test our solution in less conventional maps.

3.2 Solution Instance Generation

To generate solutions for our dataset, we initially create tours based on three different heuristics: random, *2-opt* and *nearest neighbour* heuristics. Randomly generated tours have no regards for the quality of the solution, and a simple random distribution of the cities is used to generate one tour; With the nearest neighbour heuristic, initially a city is chosen at random and the nearest city is chosen as its neighbour, and removed from the pool of available cities. This cycle is repeated until cities are no longer available and the entire tour is generated. Finally, 2-opt generated tours use the complete 2-opt method extensively, where a random tour is given to the algorithm, and then a slice is inverted. If the new tour is an improvement the result is kept and the algorithm is applied to the new tour. Because of computational constraints, the algorithm has a maximum of 1 million improvements, after which point it returns the best found tour.

At the end of this step, 1000 tours are generated with each heuristic, in each map, which means a total of 9000 initial examples are created at the start. This was done in order to have a balanced dataset with a large range of solution qualities to be able obtain as much information as possible from the generated solutions.

Afterwards, we take each of these solutions and perform a small **random walk** by applying the three operators described in Section 3.3 to each. All operators are always applied to any given solution, to two random cities in the permutation, branching it out as a tree with three new tours generated from the original solution. This process is done iteratively until each tour has generated a tree with eight levels of depth, and the leaves of said tree are saved, alongside the metrics mentioned in Section 3.4, as examples to be used on the final dataset. Figure 3.2 shows an overview of how the random walk every solution performs is done, and where the final examples are extracted from.

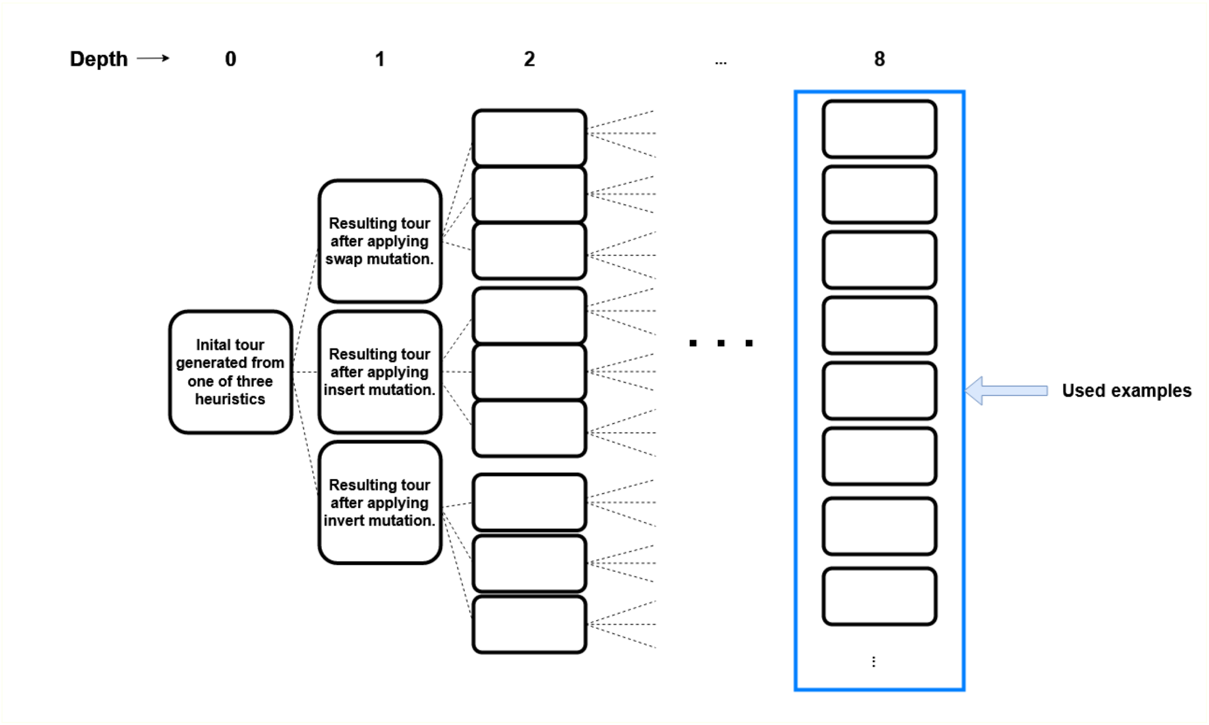

Depth
0
1
2
...
8
Inital tour generated from one of three heuristics
Resulting tour after applying swap mutation.
Resulting tour after applying insert mutation.
Resulting tour after applying invert mutation.
Used examples

Figure 3.2: An overview of the how the random walk on every generated solution is performed.

The possibility of using examples from not only the tree "leaves" but also from different depth levels was discussed, but after a brief exploratory analysis done to compare both the inner levels and the leaves we concluded that the impact these examples have on the results is negligible and therefore decided to focus only on the latter.

The resulting dataset consists of a total of roughly 5 900 000 generated examples, across the three different maps, each with 25 features available. It is an extremely large dataset and as such, very computationally expensive to study. Therefore, the decision was made to only use about 300 000 examples from each map, approximating 900 000 examples for experimentation and analysis of results. We find this is an appropriately large number of solutions to achieve consistent results, while easing the load of exploring the entirety of the available data at once.

3.3 Selected Operators

Swap Operator

A simple operation where we take two randomly selected cities in the permutation and swap their position. This operator breaks four city connections and, as such, is the most destructive operator amongst the selected three as it changes the most city connections out of the three chosen operators (see Fig. 3.3 for an example).

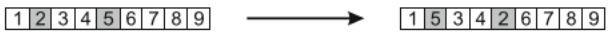


Figure 3.3: Swap operator as seen in [13].

Insertion Operator

An insertion operator selects two random cities in the tour and puts the second city next to the city that was selected first and the other values in-between are "pushed" to the side (see Fig. 3.4), breaking three city connections.

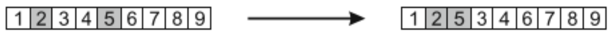


Figure 3.4: Insert operator as seen in [13].

Inversion Operator

Finally, with an inversion operator we select two random cities again, and everything between said cities (including the chosen) are inverted in order. While it seems to be the most destructive operation at first glance, it realistically only changes two links in the cities, and as such is considered the smallest change to be made in an adjacency-based problem [13].

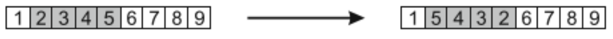


Figure 3.5: Inversion operator as seen in [13].

3.4 Metrics

As previously mentioned, this work seeks to utilize fitness landscape related metrics, and as such, our focus when creating the dataset is to retrieve information based on these metrics. This section is dedicated to detailing each of the features included in every example of the dataset.

Fitness

The fitness function measures the quality of a given solution. In our work, we consider the euclidean distance between city pairs as the fitness function of a generated instance, which means that, the lower the distance, the better the quality of a tour and as such, lower fitness values directly correlate to a smaller tour.

By considering an instance x an ordered list of city indexes $[x_0, x_1,..., x_n]$ of length *n+1*, each with coordinates $x_{n_{xy}}$, then the fitness function $f(x)$ can be described by equation 3.1.

$$f(x) = (\sum_{i=0}^{n-1} \sqrt{(x_{i+1_x} - x_{i_x})^2 + (x_{i+1_y} - x_{i_y})^2}) + \sqrt{(x_{0_x} - x_{n_x})^2 + (x_{0_y} - x_{n_y})^2} \tag{3.1}$$

As we can see, the euclidean distance is calculated for every pair, with the last city connecting to the first one in the list.

Fitness Variation

In addition to fitness, we also keep track of the fitness variation at every step the solution takes throughout the random walk, by registering the change in fitness from the last step to the new one after applying an operator to it, as can be seen in equation 3.2.

$$\Delta f_i = f_i - f_{\textit{i-1}} \tag{3.2}$$

Distance to Optimum

The distance to optimum metric chosen is the Hamming distance, obtained by measuring the distance d of the adjacency matrix generated from a given tour to the adjacency matrix generated by the optimal solution. The adjacency matrix A_{ij} is a square nxn matrix of zeroes, where an element A_{ij} on each row takes the value of one, meaning city i has an edge that connects to city *j*. The distance is then measured by the number of rows that are the same in both matrices, with the best value being zero, indicating a solution that is exactly the same as the optimum, and the worst value being n, indicating a solution without any city pair in common.

Fitness Distance Correlation

Fitness Distance Correlation (FDC) is a way of assessing the difficulty of a problem, by measuring the relation between fitness value and distance to the optimum of the search space. The FDC was first described by Jones *et. al* [19] and is defined as ϱ and described as seen in Equation 3.3.

$$\varrho(f, d) \approx \frac{1}{\sigma_f \sigma_d m} \sum_{i=1}^{m} (f_i - \bar{f})(d_i - \bar{d}) \tag{3.3}$$

Where, m is the length of the random walk at the given solution, f_i is the fitness of solution i, d_i is the distance to the optimum of solution i, $\bar{f}$ and $\bar{d}$ are the average fitness and distance to optimum across the whole random walk, and σ_f and σ_d the standard deviation of the fitness and distance, respectively. Because we are in the context of a minimization problem, when ϱ is close to 1.0, it indicates a strong correlation and as such, an easier search space. On the other end, a value of ϱ = - 1.0 indicates little correlation, and a harder landscape, suggesting a more difficult search [19].

Autocorrelation Function

The autocorrelation function gives us the structure of the landscape by measuring the degree of correlation between points in a random walk. The autocorrelation function, first coined by Weinberger *et al.* [45], is defined as $\rho(s)$, and is described by Equation 3.4,

$$\rho(s) \approx \frac{1}{\sigma_f^2(m-s)} \sum_{i=1}^{m-s} (f(i) - \bar{f})(f(i+s) - \bar{f}) \tag{3.4}$$

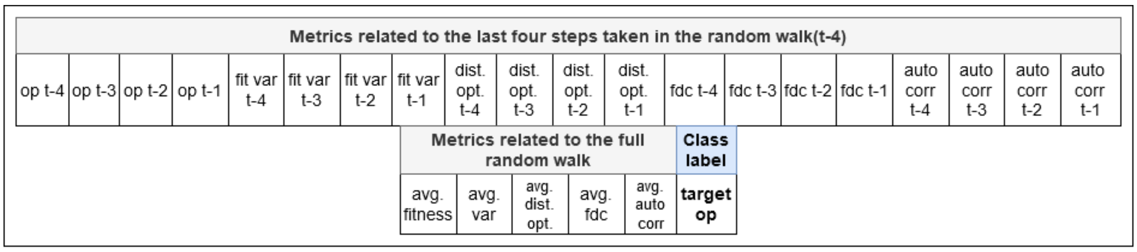
which defines the correlation of two points s steps away from each other on a given m-long random walk. The bigger the difference in fitness, the less correlation there is in the landscape and the harder it is to search for a solution. For the results we present, we consider s=1, or, in other words, the adjacent solutions in a given random walk.

Operators

In addition to the previously mentioned metrics, we include as a feature the previous operators used when generating a random walk. Any given example has the previous four operators that were applied applied during the walk featured as a metric. See section 3.3 for an in-depth description of the used operators.

Local Exploration versus Global View

An important topic that is also worthy of analysis is the importance of metrics global to the path the generated tours took while exploring the landscape, and the more local information that can be obtained from the walk. As such, the features that were extracted from the results can be separated into two categories: features that relate to the **local state** of the walk, where we can retrieve local information up to *four steps back*; and information related to the **general state** of the walk, which contains metrics pertaining to the overall performance of the walk taken by the tour. Table 3.1 gives an overview of the features included in each extracted example, including their name, and whether they contain continuous values or are categorical variables, and Figure 3.6 shows what any example in the generated datasets looks like. *Op* stands for operator used, *fit var* stands for fitness variation metrics, *dist. opt.* refers to distance to the optimum, *fdc* is related to fitness distance correlation measures, and *auto corr* describes autocorrelation values. *T - x* refers to the step in time the metric was taken in. For instance, *auto corr t-4* stands for the autocorrelation value the example displayed four steps ago on the random walk.



Metrics related to the last four steps taken in the random walk(t-4)																			
op t-4	op t-3	op t-2	op t-1	fit var t-4	fit var t-3	fit var t-2	fit var t-1	dist. opt. t-4	dist. opt. t-3	dist. opt. t-2	dist. opt. t-1	fdc t-4	fdc t-3	fdc t-2	fdc t-1	auto corr t-4	auto corr t-3	auto corr t-2	auto corr t-1

Metrics related to the full random walk					**Class label**
avg. fitness	avg. var	avg. dist. opt.	avg. fdc	avg. auto corr	**target op**

Figure 3.6: All metrics present on any given instance found in the dataset.

Local Metrics (4 Step Window)		Global Metrics (averages)		Target	
Name	Type	Name	Type	Name	Type
used operator	category	avg. fitness	continuous		
fitness variation	continuous	avg. variation	continuous		
dist. to opt.	continuous	avg. dist. opt.	continuous	target operator	category
fdc	continuous	avg. fdc	continuous		
autocorrelation	continuous	avg. autocorrelation	continuous		

Table 3.1: Overview of available features for every example. Note that every local metric includes four features, as up to four steps of the solution are available information.

Labeling

Lastly, because we decided on a supervised learning approach to classification, all the generated solutions were labeled with the optimal target. This target is decided by applying each of the three available operators in a random tour once, and choosing as the label **the operator that most decreases the fitness of the solution**, or, at the very least, is the less destructive on the quality of the solution. A display of the metrics present on a generated solution can be found on Table 3.1, showing all the collected metrics and label found on every generated solution found in the dataset.

An in-depth analysis of the target distribution results is found in the next chapter, as well as the detailing of the exploratory analysis done on the extracted data, including studies on the importance of each feature and observation of classification results using decision trees.

This page is intentionally left blank.

Chapter 4

Experimental Study

This chapter is dedicated to the exploration of the data obtained in the previous chapter. We start by presenting the summary of the statistics obtained from the observation of the different metrics extracted from the generated solutions on Section 4.1, studying each feature and their relation with one another, then drawing conclusions from said analysis. Observations such as the distribution of targets, and the relevance of each feature for the building of classifiers are also found here. Following that, we summarize the results obtained in the classification tasks in Section 4.2, and then compare our model against a regular hill climber in Section 4.3.

4.1 Exploratory Data Analysis

Here, we present the results of the analysis of the obtained metrics, and draw conclusions from the generated examples. For that effect, we performed univariate analysis on each observed feature to find patterns and describe the behavior of the found results. We then try to understand the relation between different features and how they interact with one another, as to gain insight into the relevance of different features, and highlight their usefulness in selecting different operators.

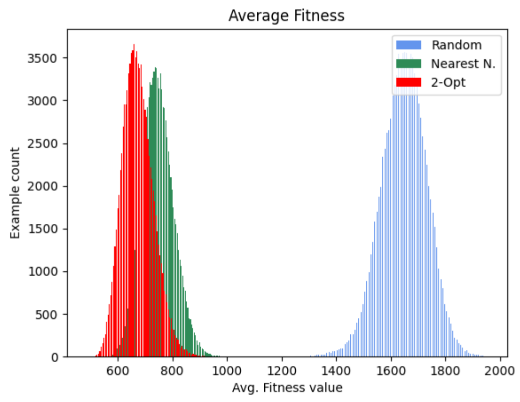
4.1.1 Univariate Analysis

In this section, we present the results of observing each feature's behavior across the different instances, aiming to discuss emerging patterns found across the different metrics.

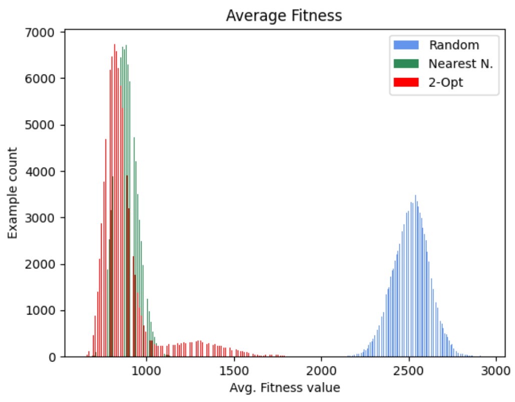
Fitness

Figure 4.1 displays the distribution of fitness from example solutions extracted from the dataset, according to their average fitness value.

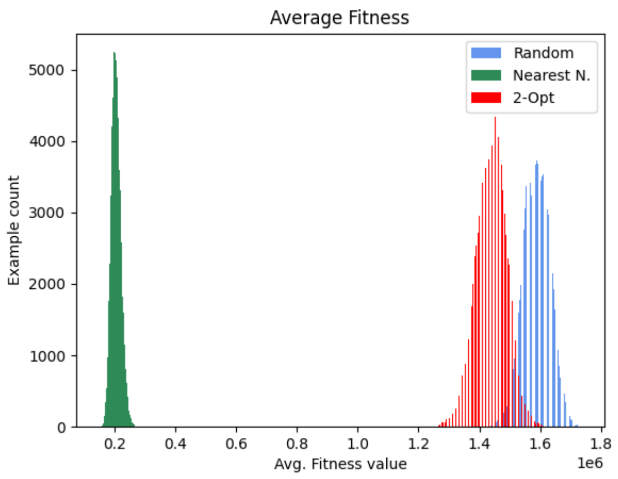
The observed fitness values show that 2-opt and nearest neighbour tours present generally better quality solutions, while random tours have notably worse fitness values. An exception to this is the 225 city map, where 2-opt generated tours are only a slight improvement over randomly generated tours. This is a common pattern found in the data, which is to be expected, as this map in particular is purposefully a difficult map to solve, due to the combination of a higher number of city pairs than the other two, and due to the deceptive



(a) Avg. fitness example count for *eil51*.



(b) Avg. fitness example count for *eil76*.



(c) Avg. fitness example count for *ts225*.

Figure 4.1: Average fitness distribution across generated solutions on the three different maps, with generation heuristics highlighted - blue represents randomly generated solutions, red represents 2-opt generated solutions, and green represents nearest neighbour solutions.

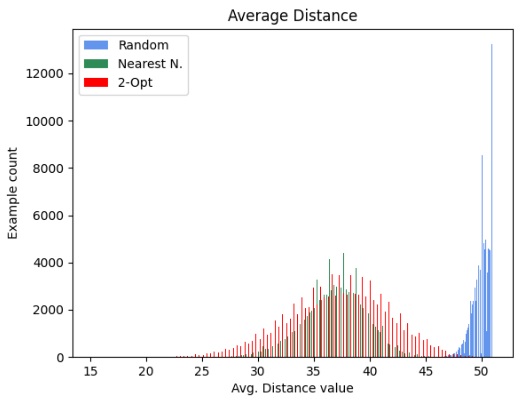
difficulty inherent to the map [29]. This fact, alongside the fact that the number of improvements applied to 2-opt tours is limited to a million per tour, contribute to the results displayed in Fig. 4.1c.

Distance to Optimum

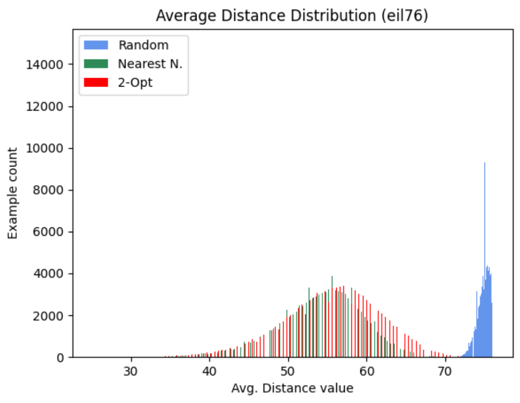
The average distance to the optimum across the different map instances is displayed in Figure 4.2, and is within the expected values - as observed above, tours generated with nearest neighbour and 2-opt exhibit similar behavior and present better results overall, with 2-opt tours having a more even distribution of values (except for the 225 city map, as discussed above).

The presented Figures 4.2 and 4.1 hint at an existing correlation between distance and fitness, as the observable results parallel each other when directly compared. However, it is important to note that the distribution of both metrics does not entirely overlap (for example, Fig. 4.2c presents a wide range of solution distribution according to distance to optimum on nearest neighbour generated tours, while Fig.4.1c shows the same solutions presenting a similar fitness value), which exemplifies that just because an solution's fitness is improving, it does not directly correlate to a convergence to the optimal solution, which leads us to conclude that we cannot solely rely on fitness to look for the best possible operator.

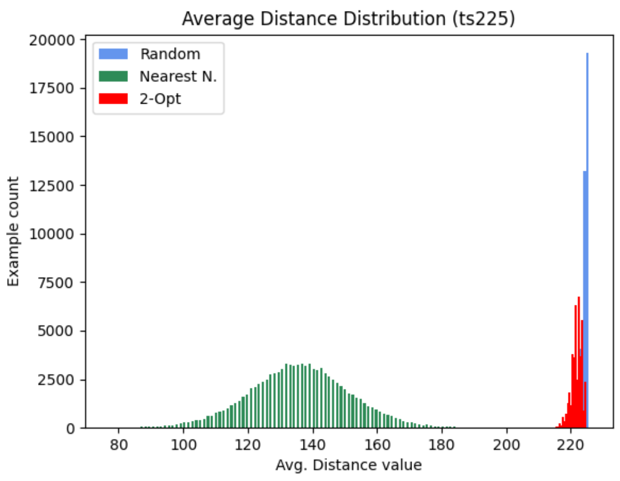
The two shown metrics together however, display a common pattern when it comes to solution quality, which we will assume true from this point forward - randomly generated tours are generally worse initial solutions, while nearest neighbour and 2-opt tours (on the 51 and 76 city maps, at least) should be considered better quality initial solutions.



(a) Avg. distance example distribution for *eil51*.



(b) Avg. distance example distribution for *eil76*.

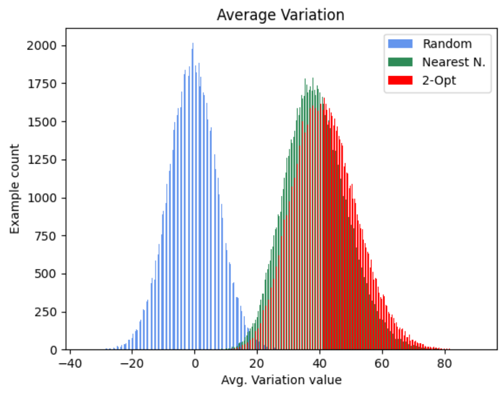


(c) Avg. distance example distribution for *ts225*.

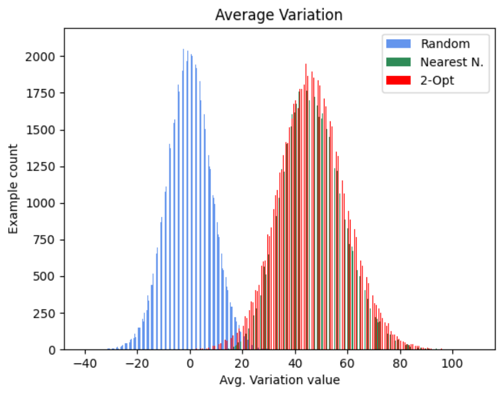
Figure 4.2: Average distance to optimum across different maps, with different generation heuristics highlighted - blue represents randomly generated instances, red represents 2-opt generated examples, and green represents nearest neighbour instances.

Fitness Variation

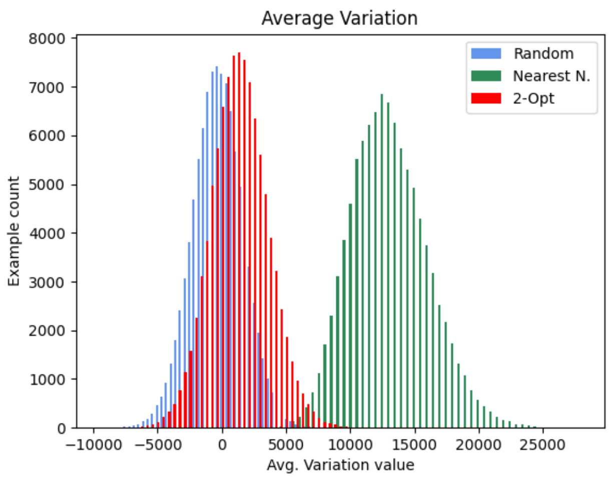
From the fitness variation plots found in Fig.4.3, we can observe that most operations applied to already good solutions are detrimental to their quality, and mutations applied to randomly generated tours seem to be evenly distributed in positive/negative changes. Another observation worthy of note is that, as expected, the swings in fitness quality are much steeper as the number of cities increases, as per Figure 4.3c.



(a) Avg. fitness variation example count for *eil51*.



(b) Avg. fitness variation example count for *eil76*.



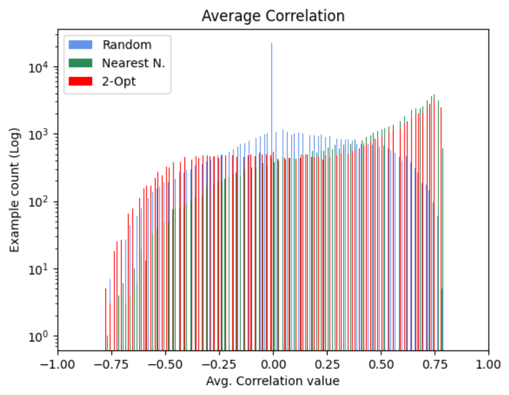
(c) Avg. fitness variation example count for *ts225*.

Figure 4.3: Average fitness variation distribution across different maps, with different generation heuristics highlighted - blue represents randomly generated instances, red represents 2-opt generated examples, and green represents nearest neighbour instances.

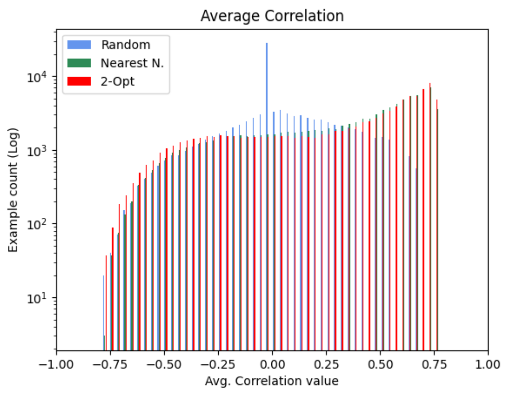
As such, we can conclude that fitness variation can aid in determining the quality of the tour when looking at its average value, and seems to be especially useful when combined with distance and fitness measures to find out the appropriate operator to be applied on a given solution.

Fitness Distance Correlation

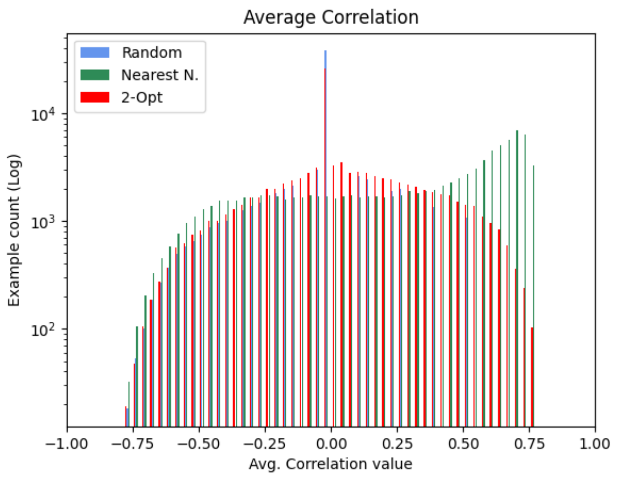
The plots presented in Figure 4.4 tell us that the vast majority of solution with a random initialization has a fitness distance correlation value of near zero, which suggests that most of the randomly generated examples does not present significant variation on fitness and distance values across their random walk, based on Equation 3.3, which indicates a difficult search for the optimum on these solutions. Additionally, the 2-opt and nearest neighbour generated tours, present a similar pattern on correlation distribution, with very disperse values across the whole dataset, with a slight tendency to increase to a peak of counts at the 0.6 to 0.8 interval, which would indicate a stronger correlation between fitness and distance on these solutions, and an easier search for the optimum.



(a) Avg. fitness distance correlation example count for *eil51*.



(b) Avg. fitness distance correlation example count for *eil76*.



(c) Avg. fitness distance correlation example count for *ts225*.

Figure 4.4: Average fitness distance correlation value distribution across different map instances, with different generation heuristics highlighted - blue represents randomly generated solutions, red represents 2-opt generated solutions, and green represents nearest neighbour solutions.

The progression of FDC values across the last four steps of the random walk of the generated solutions can further be evaluated in Fig. 4.5. Of particular note here is the reduction of examples with a FDC value of zero as the time window progresses towards the current solution, and the bigger spread of fitness distance correlation values observed as a result. This is beneficial, as more variety in the examples of the dataset leads to more information gathered from the available metrics, and a more informed decision can be made by our ML model to select an operator. However, the large spread of featured values of FDC across the generated solutions indicates that the degree of difficulty in the search for an optimum is highly dependant on where the solution is situated on the search space, suggesting an uneven landscape across the different maps, and as such, might be an unreliable metric for operator prediction in our work.

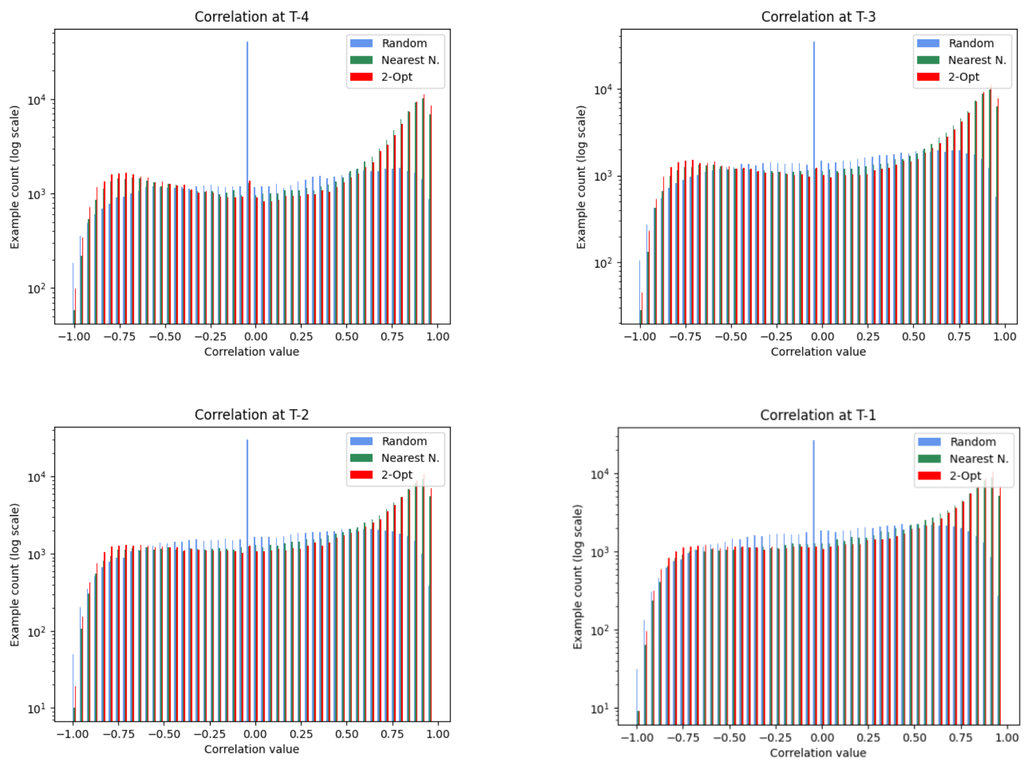


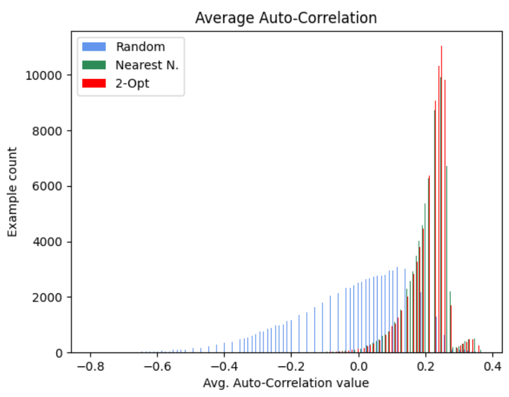
Figure 4.5: Progression of the local fitness distance correlation value across examples on the *eil76* map.

Autocorrelation

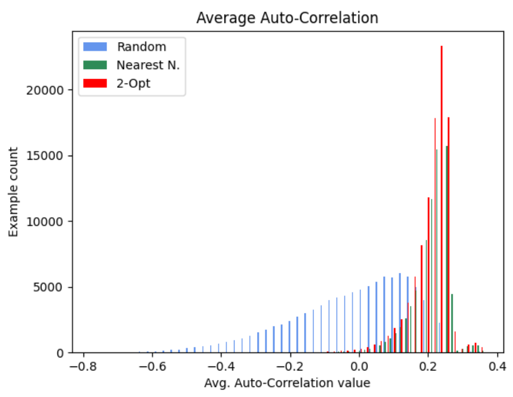
From the observation of the autocorrelation results, some conclusions can be drawn. Worse initial solutions seem to be dispersed across a wide range of values, as evidenced by Fig. 4.6, and furthermore, tend to have, on average, lower values. The opposite is verified on the majority of higher quality examples, which tend to not only feature a much higher example count of solutions with higher values of average autocorrelation, but also a much smaller range of values. This tells us that higher autocorrelation values are generally most likely to be tied to higher quality examples. Additionally, the progression of the autocorrelation values over the four-step window observed in Fig. 4.7 shows that as the walk is performed, the count of solutions with higher autocorrelation also increases, further converging in the upper end of possible values, which reinforces that better initial solutions are more likely to also have higher values of autocorrelation. Autocorrelation directly ties with the landscape properties of the problem instance - higher autocorrelation values translate to a more rugged landscape, and a harder search for the solution [41], and therefore, the results suggest the search is increasingly harder as we get closer to the optimum, as weaker solutions display lower values of autocorrelation, and solutions with better fitness/distance to optimum present higher autocorrelation metrics.

Operator Usage

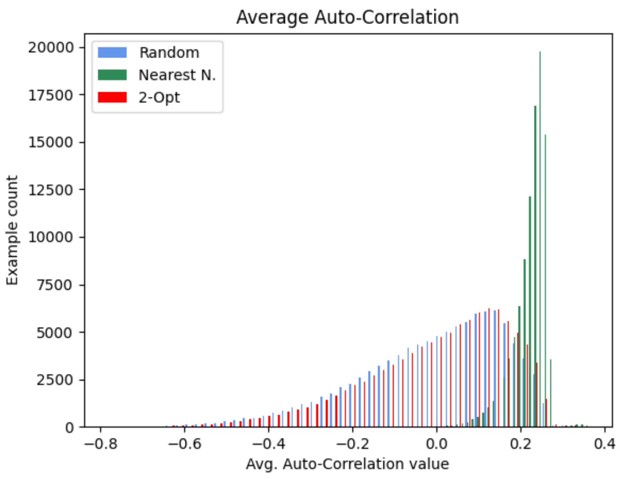
As mentioned in section 3.2l, the dataset features an extremely large number of generated solutions. Because we do not work with all the examples in the dataset as that would require more computational resources than what was available, we use a subset of solutions



(a) Avg. autocorrelation value example count for *eil51*.



(b) Avg. autocorrelation value example count for *eil76*.



(c) Avg. autocorrelation value example count for*ts225*.

Figure 4.6: Average autocorrelation distribution across generated examples on the three different maps, with generation heuristics highlighted - blue represents randomly generated instances, red represents 2-opt generated examples, and green represents nearest neighbour instances.

extracted from each different map instance to work on. Therefore, an analysis to determine how many times each operator is used across all generated examples at all steps of the 4-step window is done, to guarantee we are working with a dataset with a balanced example count. As shown in Figure 4.8, the operator usage is practically even across all steps. While only the results for the 76 city map are displayed, all instances display similar patterns, and therefore we can safely state that the operator usage is entirely even across all examples, which guarantees we are working with data that has has an even distribution of random walk exploration using the chosen operators. Furthermore, this also shows there is little to be gained in evaluating the previous operators when selecting a new one, something that we look at more closely at in Section 4.1.4.

Label Distribution

Finally, the label class distribution plot observed in Figure 4.9 is a good indication that the operators are behaving as expected. As stated in Section 3.4, the examples are classified

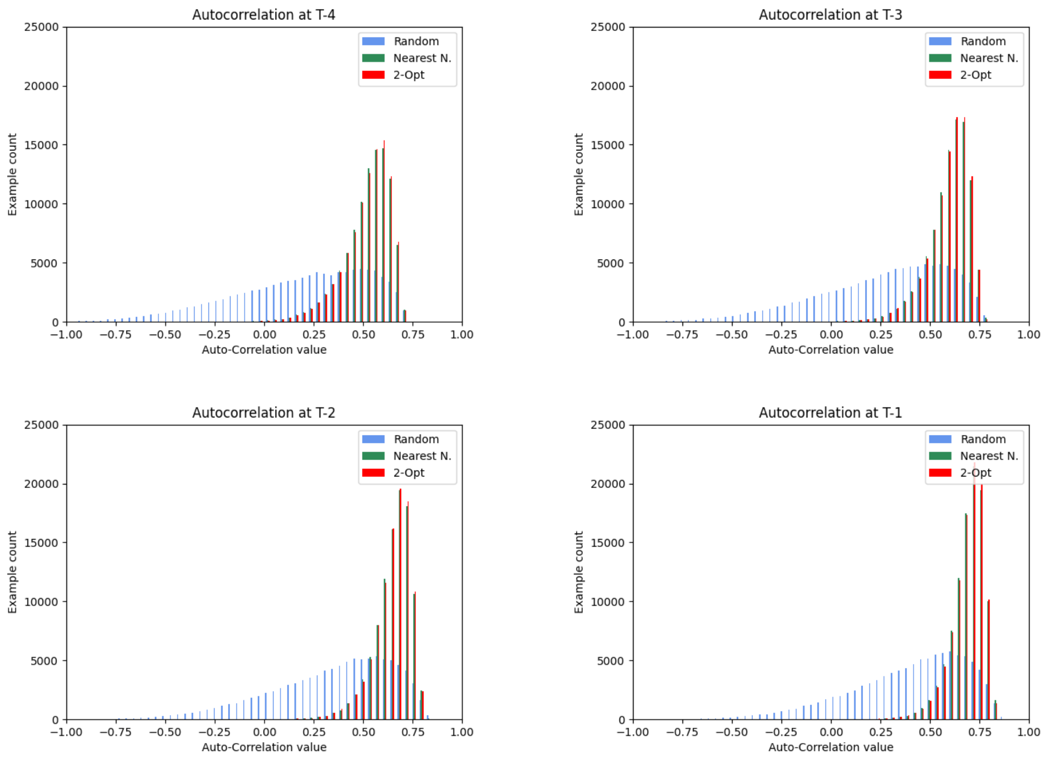


Figure 4.7: Progression of the autocorrelation values across extracted examples on the *eil51* map.

based on the application of all three operators on the given instance on two random cities, and compares the fitness variation of all three generated solutions, choosing as a label the operator that most improved (or, least damaged), the original solution. The results show that swap operator is used as a label with more frequency by randomly generated tours than by better heuristics. Interestingly enough, randomly generated solutions have an almost equal number of different operators used as a target across all maps. Nearest neighbour and 2-opt generated solutions follow a similar pattern of target selection across instances, with insert and invert operators being used more frequently when the heuristic for original tour generation has better quality, which is expected behavior, except at the 225 city map, where we can observe that 2-opt tours follow the behavior observed in random tours, as expected from the quality of the tours generated with that heuristic on said map.

In short, swap operator labeling is found in near even number across all examples and maps, while inversion and insert operators are used much more commonly on better quality heuristics, with these having a smaller amount of solutions labeled with swap operator overall. A more extensive analysis of target distribution can be found in Section 4.1.3.

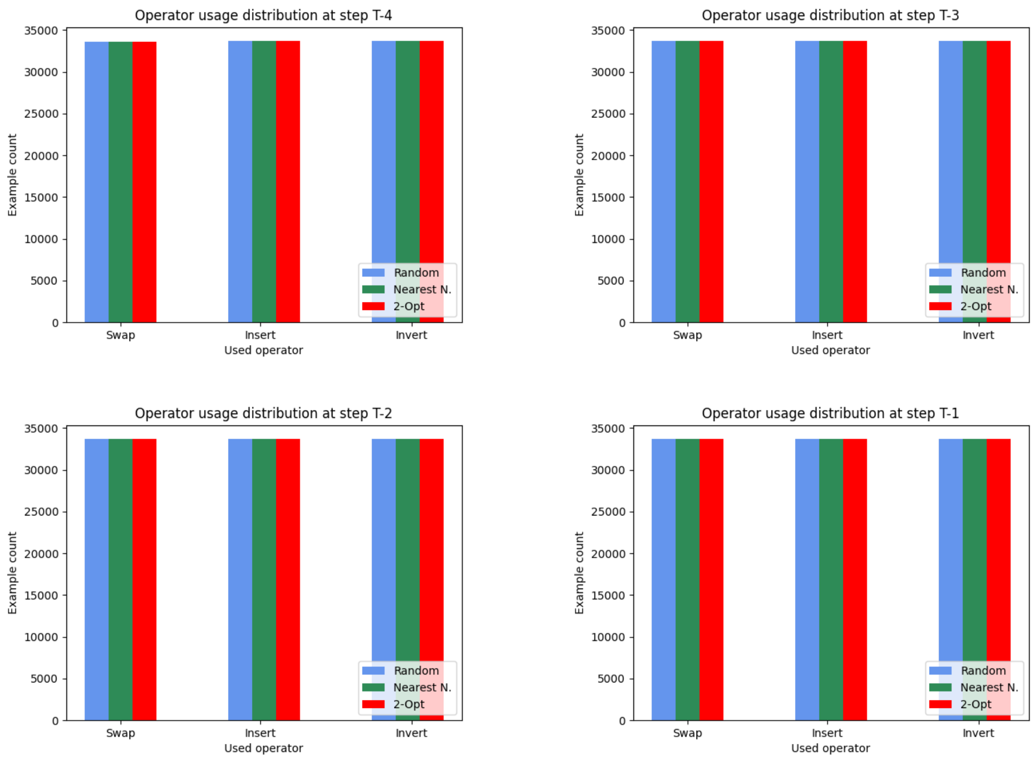
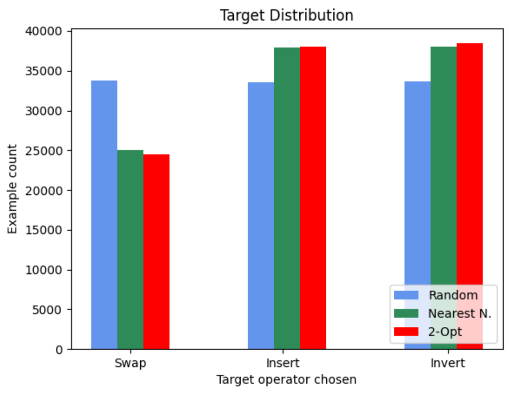
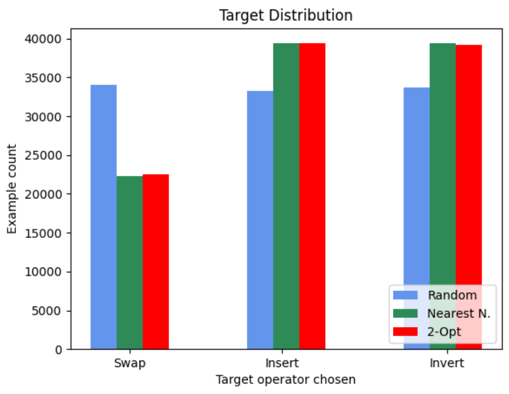


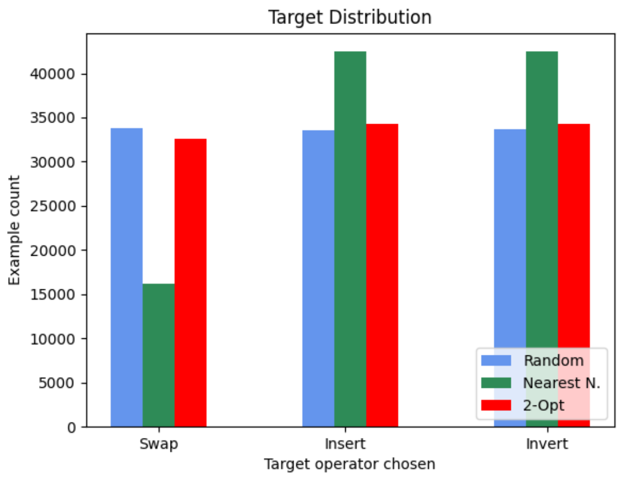
Figure 4.8: Distribution of the three different operators selected across the 4-step window on the *eil76* map.



(a) Labeled class example count for *eil51*.



(b) Labeled class example count for *eil76*.



(c) Labeled class example count for *ts225*.

Figure 4.9: The labeled examples distribution across all maps, according to the heuristic used to generate the original solution.

4.1.2 Multivariate Analysis

In this section we analyse the different metrics in relation to one another, in order to try and find patterns and relations that are not apparent at a first glance. Note that for the most part, nearest neighbour generated examples (marked green on the scatterplots) are barely visible, mostly due to their similar behavior to 2-opt generated examples (in red), meaning that in most cases they are simply hidden behind the latter.

Autocorrelation And Fitness Distance Correlation

Figure 4.10 tracks the evolution of the FDC/autocorrelation values on the four-step window. What is interesting to observe here is how noticeable the convergence of higher quality solutions to higher values of autocorrelation is compared to randomnly generated solutions. Other than that, the change in FDC values seems to be more negligible, with only a slight increase in their values over time.

Figure 4.11 presents the scatterplots for the average values of both metrics in a single map, with similar results to what was gathered before. It is apparent there is no real correlation between the two features, as the displayed values are very disparate: average FDC values are very scattered across the graphs, while average autocorrelation values tend to be higher the better the solution quality, as was observed previously in Fig. 4.6.

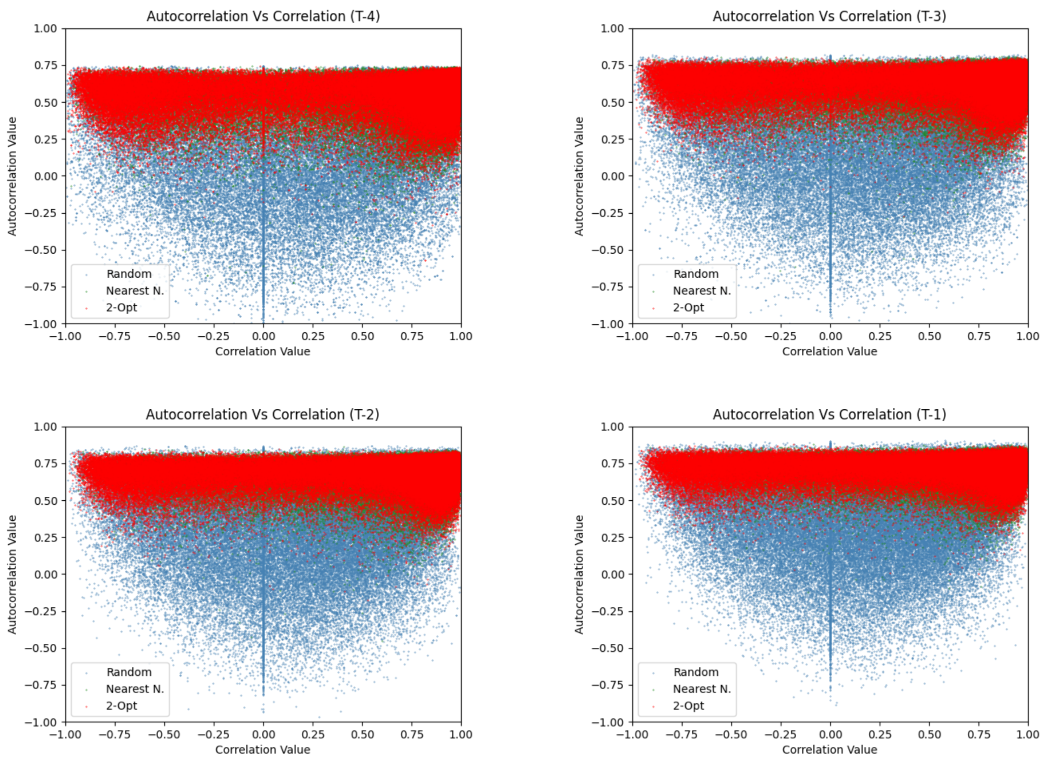


Figure 4.10: Scatterplot of the relation between autocorrelation values (y axis) and FDC values (x axis) across the 4-step window on the *eil51* map.

For both figures only the results for one map are displayed, as all feature similar patterns in data distribution.

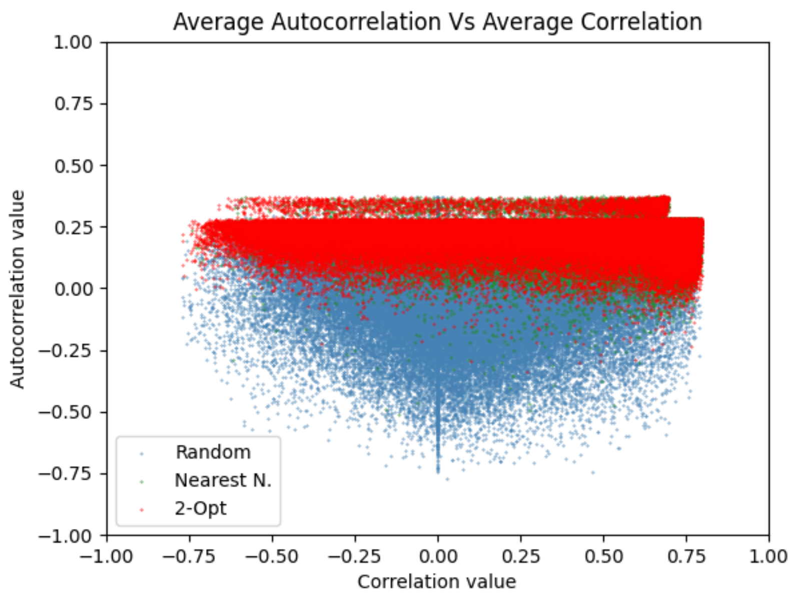


Figure 4.11: Scatterplot of the average autocorrelation values (y axis) and the average FDC values (x axis) found in *eil51* generated solutions.

Distance to Optimum and Autocorrelation

The scatterplots comparing these two metrics are found in Figure 4.12. From here, we can conclude that distance to optimum and autocorrelation are tied to one another to some degree. While it is true that examples that are further away from the optimum present big disparity on autocorrelation results, it seems that as a general rule, tours that present shorter distances to the optimal solution generally display higher values of average autocorrelation.

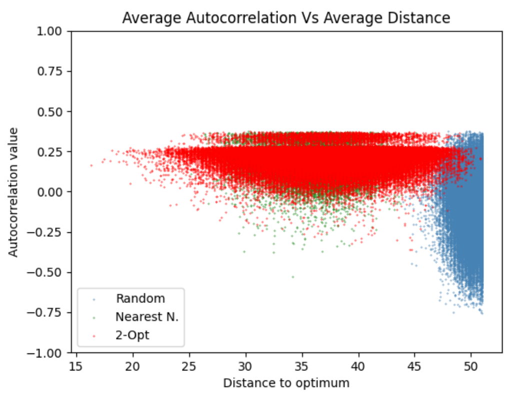
Distance to Optimum and Fitness

The scatterplots shown in Figure 4.13 further strengthen the hypothesis that fitness and distance are not linearly correlated - just because a solution displays lower fitness values it does not equate to a solution that is close to the optimum. However, despite this, there are no solutions with low distance to the optimum and high fitness, which means that, while not directly tied to one another, one can draw the conclusion that better solutions always have better fitness values.

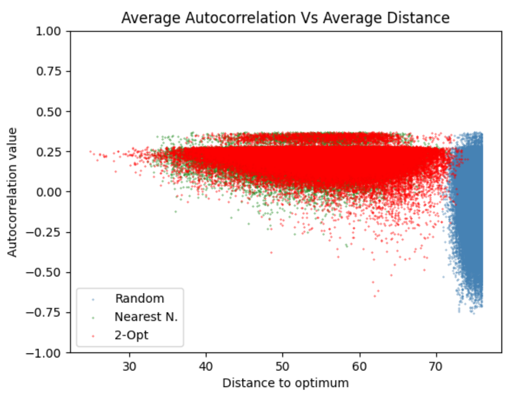
Another interesting observation to be had, is that the 2-opt generated tours seem to have a higher disparity of fitness in relation to the distance to the optimum, especially in the 76 city map. This is a good indication of how fitness can be a deceptive measure on its own, as higher fitness values can sometimes equate to better quality solutions too, and as such is important to not rely solely on it as a metric for solution quality.

Fitness Variation and Fitness

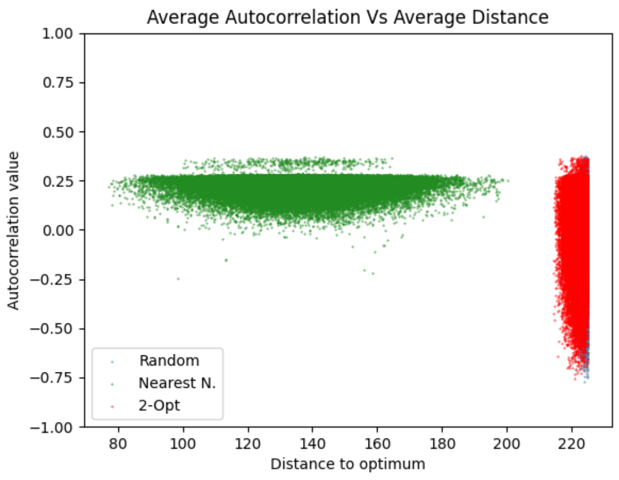
The plots displayed in Figure 4.14 show what is to be expected. Tours generated with better heuristics have overwhelmingly worse average variation due to the operators applied



(a) Average autocorrelation/distance to optimum scatterplot for *eil51*.



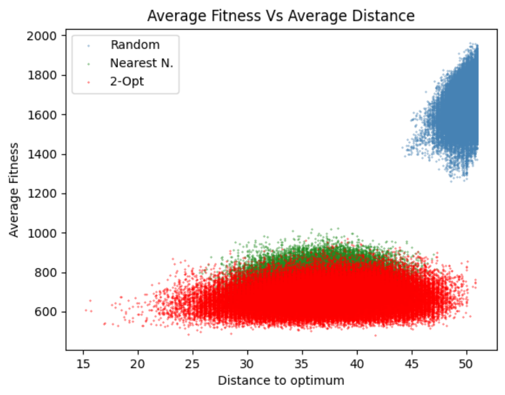
(b) Average autocorrelation/distance to optimum scatterplot for *eil76*.



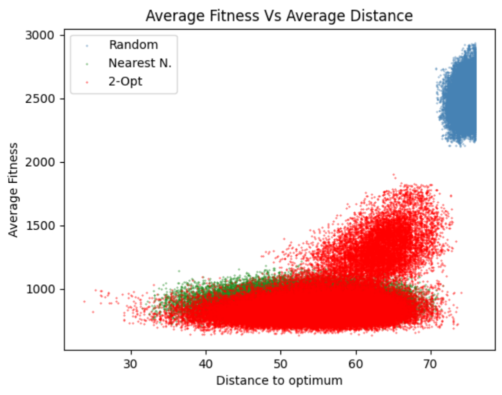
(c) Average autocorrelation/distance to optimum scatterplot for *ts225*.

Figure 4.12: Scatterplot for the average autocorrelation values (y axis) in relation to the average distance to optimum (x axis) for the *eil51* map (4.12a), *eil76* map (4.12b) and *ts225* map (4.12c).

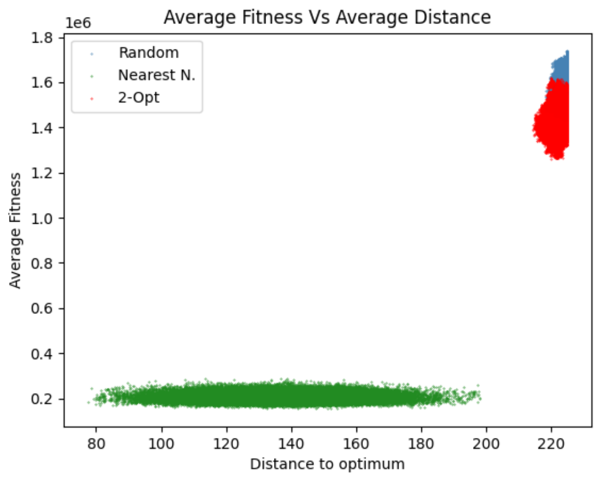
not being discriminatory. Also as expected, the tours with the highest fitness also tend to improve slightly or not worsen as much as others. Also noticeable, is how fitness variation is observably tied to the fitness of a solution, and furthermore, the number of cities in the map and the tour length both affect the swings in fitness, with bigger average fitness variation value changes from one solution to another displayed as the cities in the map increase. This also ties back to autocorrelation values being generally higher on solutions with a higher fitness quality, with the bigger swings in fitness reflecting the increased jumps in fitness across a random walk and increase in autocorrelation as a consequence.



(a) Average fitness/distance to optimum scatterplot for *eil51*.

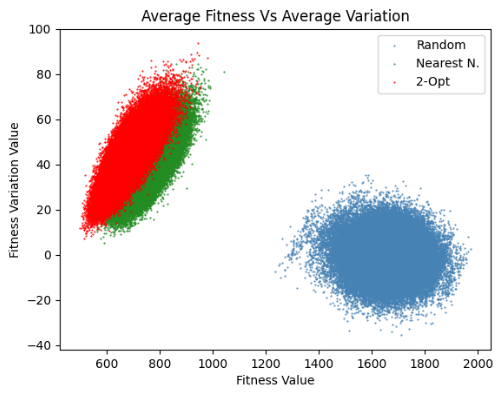


(b) Average fitness/distance to optimum scatterplot for *eil76*.

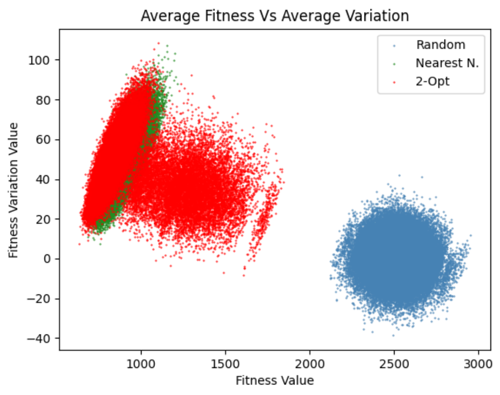


(c) Average fitness/distance to optimum scatterplot for *ts225*.

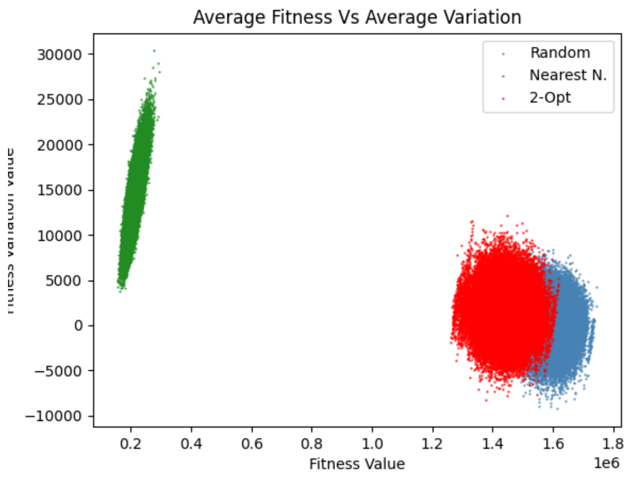
Figure 4.13: Scatterplot for the average fitness values (y axis) in relation to the average distance to optimum (x axis) for the *eil51* map (4.13a), *eil76* map (4.13b) and *ts225* map (4.13c).



(a) Average fitness/fitness variation scatterplot for *eil51*.



(b) Average fitness/fitness variation scatterplot for *eil76*.



(c) Average fitness/fitness variation scatterplot for *ts225*.

Figure 4.14: Scatterplot for the average fitness values (y axis) in relation to the average fitness variation values (x axis) for the *eil51* map (4.14a), *eil76* map (4.14b) and *ts225* map (4.14c).

4.1.3 Target Distribution Analysis

This section is dedicated to observing how the different features interact with the operators used as labels. Through it we will explore the distribution of values among the different features while keeping in mind the target operator choice of the generated tours, instead of the heuristic used to generate the initial instances, to find patterns and information about which operator is preferred and under what conditions.

Targets According to Distance to Optimum

This visualization of target distribution present across distance in Figure 4.15 gives us further glimpses at what was already hypothesized above: as distance to optimum decreases, the preference tends towards operators that are less destructive to the tour (insert and invert), while examples with poor quality have a near even distribution of operator choice as a label.

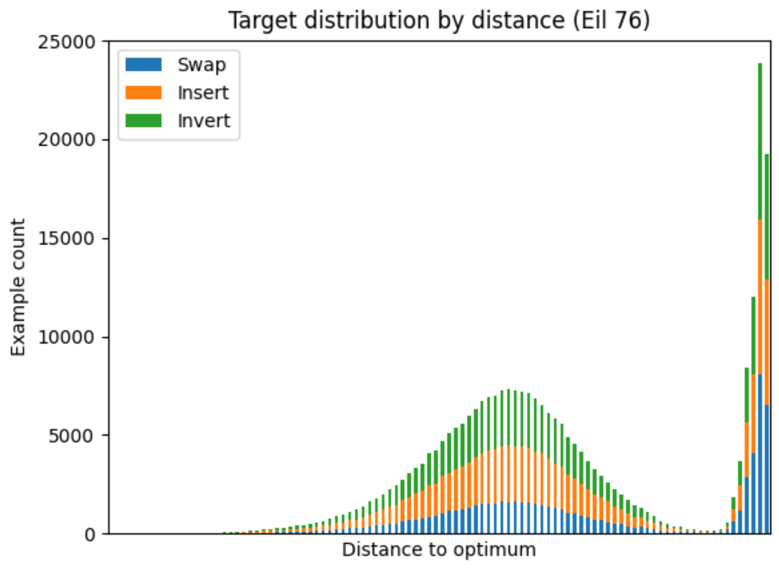


Figure 4.15: Distribution of operators chosen as labels on the *eil76* map, distributed by average distance to optimum - Swap operators are displayed in blue, insert operators are orange and invert operators are green.

Targets According to Fitness Value

The graph visualized in Figure 4.16 further corroborate what was observed in the target distribution by distance graphs - while solutions that choose swap as their target exist on some capacity on lower fitness values, the other operators are vastly preferred among the solutions of better quality.

Targets According to Autocorrelation Value

The plots displayed in Figure 4.17 further corroborate what was stated before, highlighting how evaluating the autocorrelation values can be important to determine the quality of

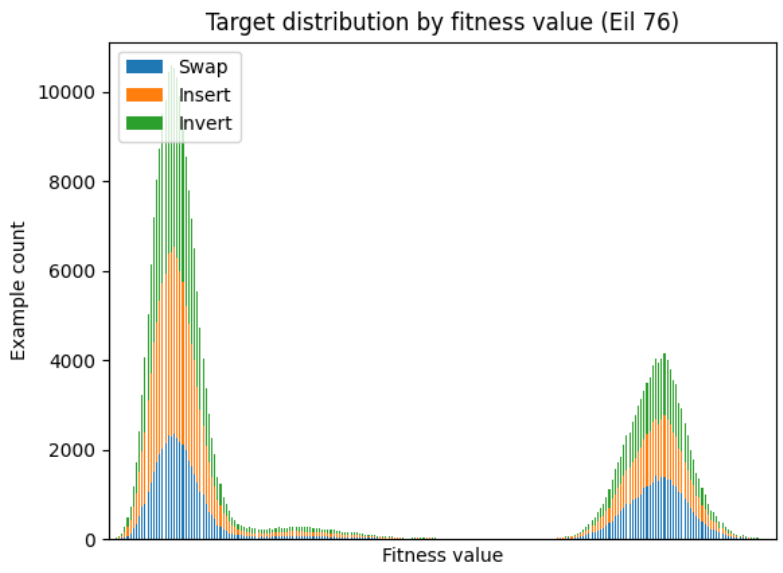
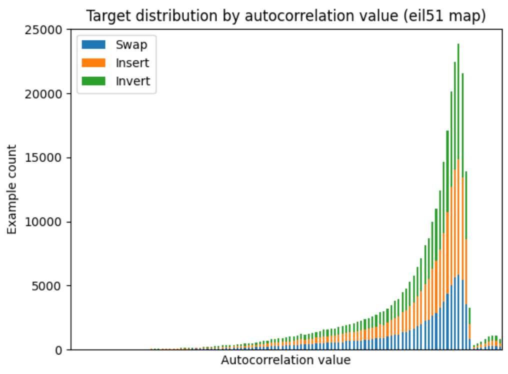
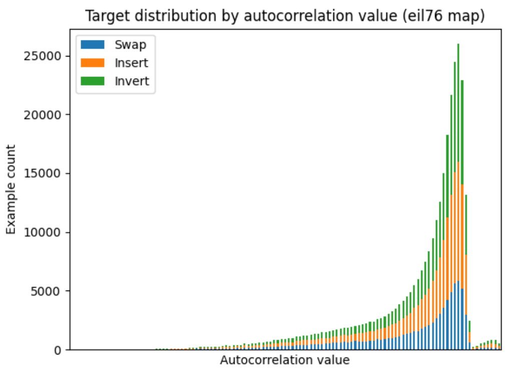


Figure 4.16: Choice operator distribution per average fitness value of all solutions from the *eil76* map - Swap operators are displayed in blue, insert operators are orange and invert operators are green.

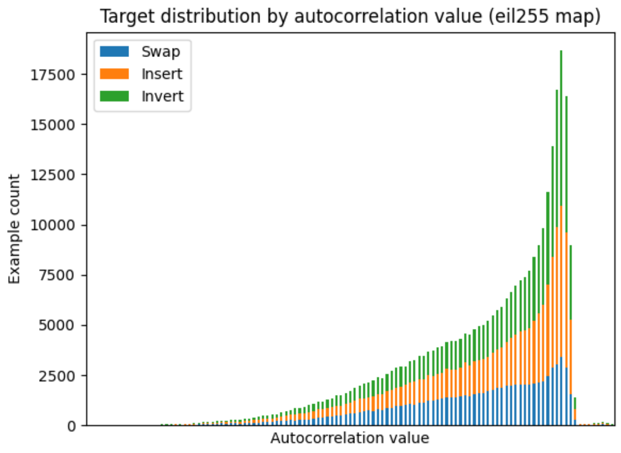
a given solution - higher autocorrelation values indeed tie to stronger solutions and thus prove to be an important metric to determine the correct operator to be chosen.



(a) Label distribution according to average autocorrelation in *eil51*.



(b) Label distribution according to average autocorrelation in *eil76*.



(c) Label distribution according to average autocorrelation in *ts225*.

Figure 4.17: Distribution of operators chosen as labels on all maps according to average autocorrelation value - Swap operators are displayed in blue, insert operators are orange and invert operators are green.

4.1.4 Feature Importance Ranking

Introduction

This subsection is dedicated to analysing the different features' importance in the task of discovering the most promising operator, and ranking them in order of their score. First we go over the chosen approach in the analysis, and then we display the obtained results.

Approach

We decided on the utilization of random forests for the analysis of the feature ranking importance. Implementation is done using sklearn's `RandomForestClassifier`. In order to test results under different conditions, we use forests with either one hundred or two hundred generated trees.

When it comes to the criteria for information gain, Entropy and Gini Impurity, they indicate roughly the same when ranking features by order of importance, and as such their difference in analysing the dataset is negligible. With that said, we choose to use Gini Impurity as our criterion.

The trees are built with default parameters, and the feature ranking analysis was performed in the dataset for the three maps separately, and for all the maps combined as well. In the combined dataset, data was normalized using sklearn's `StandardScaler` implementation, which is applied independently on each feature of each different map and then joined together for analysis. The results can be found on the section below.

Results

The obtained results can be found on Table 4.1. The feature importance analysis shows similar patterns across all maps and combinations, therefore we choose to display the results of the combination of the three maps as a representative that can be generalized to other results. The values are normalized and pertain to the total reduction of the criterion brought by that feature.

Figure 4.18 show a visualization of the values found within the table for the random forest with two hundred trees. As was previously hypothesized during the operator usage exploratory analysis, past operator usage is considerably less valuable than the rest of the features. Local distance to optimum metrics also display relatively low importance. FDC display middle of the road results. Autocorrelation related values display good results in importance on predicting a correct operator, and follow closely average fitness and fitness variation metrics, which are consistently classified as the most valued metric, displaying the overall highest scores.

		Configuration	
Feature Indexes	**Features**	**RF 100**	**RF 200**
0	Op T-4	**0.016581**	0.019445
1	Op T-3	0.016617	**0.019241**
2	Op T-2	0.016652	0.019660
3	Op T-1	0.017140	0.019657
4	Fit Var T-4	0.052538	0.051527
5	Fit Var T-3	0.052503	0.051696
6	Fit Var T-2	0.053090	0.052184
7	Fit Var T-1	0.053720	0.052620
8	Dist T-4	0.027013	0.028249
9	Dist T-3	0.025279	0.027242
10	Dist T-2	0.025155	0.027461
11	Dist T-1	0.026860	0.028775
12	FDC T-4	0.040732	0.039882
13	FDC T-3	0.040187	0.039026
14	FDC T-2	0.041019	0.039916
15	FDC T-1	0.043469	0.042323
16	A. Corr T-4	0.050867	0.050020
17	A. Corr T-3	0.050136	0.049219
18	A. Corr T-2	0.050454	0.048837
19	A. Corr T-1	0.051317	0.049671
20	Avg. Fit.	**0.056000**	**0.054583**
21	Avg. F. Var	0.053299	0.052141
22	Avg. Dist.	0.042042	0.041771
23	Avg. FDC	0.044885	0.043627
24	Avg. ACorr.	0.052446	0.051225

Table 4.1: Results for the fitness importance ranking on the combination of all maps, with different numbers of trees in the random forest (RF). Lowest and highest values on each configuration are highlighted.

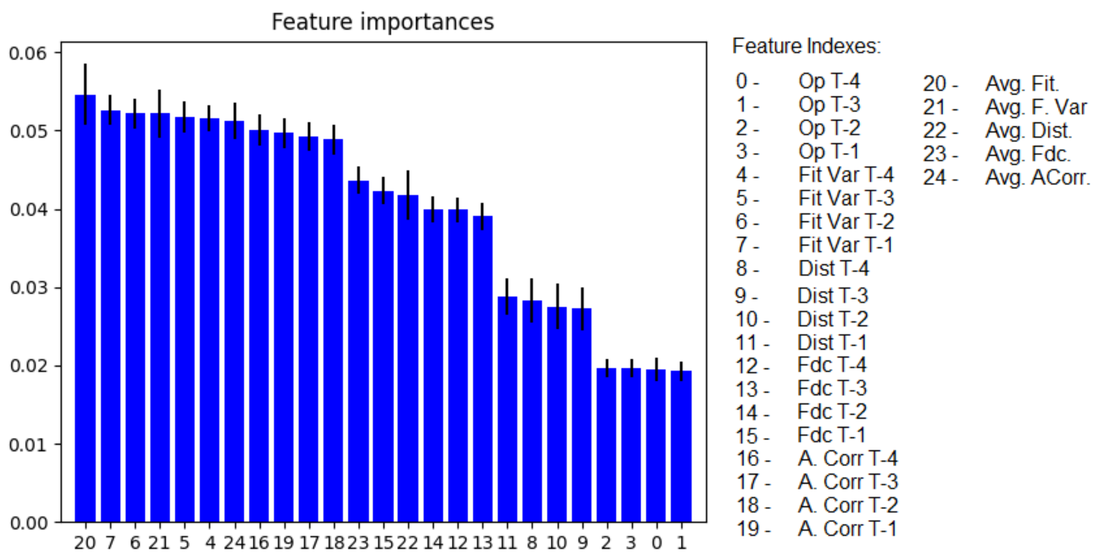


Figure 4.18: Feature ranking display for random forests with 200 decision trees.

4.1.5 Discussion

From the gathered results we can make some observations and extrapolate information pertaining to the importance of the features in the dataset. Before we move on to the classification task, it is important to draw some conclusions based both on the exploratory analysis done, and the feature classification work:

1. Previously used operators do not add to the information gained from any given example, and as such, from this point on we choose to disregard these features for the rest of the work;
2. Distance related features consistently rank on the lower half of usable metrics as well, and therefore has a lower impact on the overall information gained;
3. Fitness correlation distance metrics also rank on the lower half of the feature ranking and often present similar ranking to distance related features, which is expected due to the noticeable lack of patterns in its distribution;
4. Autocorrelation values place on the higher half of the feature ranking, which indicates that such features have relevant information to our goal and should be taken into consideration when analysing the best operator to choose. Average autocorrelation values often rank in proximity with local fitness variation values, which further corroborates the importance of said metrics;
5. Fitness variation and fitness both consistently rank at the top as the most important features to take into account, which is expected, considering they measure the quality of the solution itself and can evaluate if the operators used are straying (or not) from the correct path, as evidenced by the analysis above.

As such, with the knowledge gathered in these sections, we are now ready to discuss the obtained results below.

4.2 Results

In an early phase, several different classifiers were chosen to test the accuracy of the classification task of unlabeled instances. Preliminary tests with different classifiers,, show low accuracy values, ranging from 35% to 40%, when tasked with choosing correct operators. Because of this, we opted for using decision trees to perform an analysis of the results and understand why the accuracy scoring was low, presenting our findings on the next chapters.

4.2.1 Decision Tree Analysis

Because of the inherently deterministic nature of decision tree classifiers, the first step taken was performing initial tests with different tree depths to understand if the choice metrics on the split nodes were aligned with the feature importance metrics obtained in the previous section. To this end, we used the sklearn `DecisionTree` implementation, and tested for the feature importance metrics, with differing maximum tree depths. We used a fixed seed both in dataset sample extraction, and in tree generation in order to have reproducible results. To guarantee dataset balance when it comes to labeled data, from each map we select an even number of classified examples (about 13 000 labeled instances per class, per map), which totals a number of about 120 000 examples used on the analysis of the combined dataset.

The resulting top features used in decision nodes can be consulted in Table 4.2, for each map instance and depth. As previously stated, the featured instances do not have operator related metrics included, as previous analysis suggested they would not add value to the classification task. As we can observe from Table 4.2, in combination with the metric exploration, the most important features to be used in decision nodes coincide with the previous results for the most part. A very interesting observation, however, is how, when combining examples from all maps, the preferred features turn from average fitness and variation metrics to autocorrelation and distance metrics. This is also somewhat displayed in the *ts225* decision trees, which for lower depths prefer to use average distance as the primary discrimination factor.

We can observe an example of the built trees in Figure 4.19. Because the trees become hard to read on increasingly large depths, we opt to show a detailing of a decision tree branch as to examplify observed behavior. The coloring of the leaf nodes indicates the specificity of said node - for example, nodes with deep purple color indicate examples that fall under the specified conditions have a very high likelihood of being labeled with the "*insert*" operator class. The observation of the trees immediately root out a big problem: there is very low specificity found on many leaf nodes, which translates to difficulty in identifying the correct class, even with the most discriminatory features used to make a decision. This is something that holds true even as the depth of the tree increases. This highlights the issue that the collected metrics simply are not discriminating enough to properly label tours solely based on them, even though all metrics play a part in selecting the correct label. This is also evidenced by the very similar Gini value present on the observable nodes, revealing uncertainty in the classification of operator choice. While discrimination can and does increase the deeper the tree goes, a right balance has to be struck between depth and capability of generalization of the tree, because it can easily fall into overfitting if simply left unchecked.

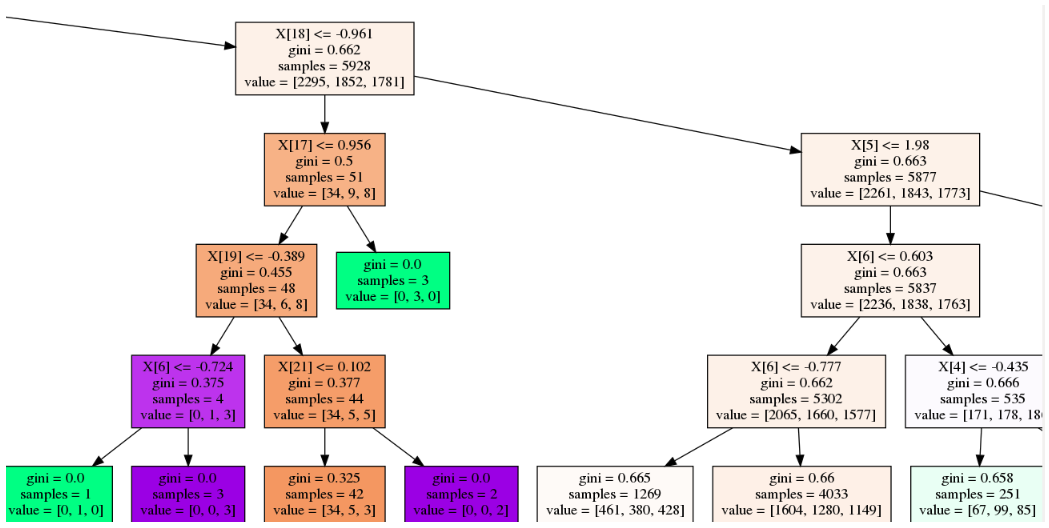


Figure 4.19: Detail of a decision tree branch of depth 8 containing examples from all maps. Note how some leaf nodes can label correctly with some certainty, but there is a lot of indecision for a great amount of samples, even as depth progresses.

This topic, alongside additional analysis, like utilizing random forests instead of single trees, is continued below. Furthermore, we compare results for scaled and unscaled data, and checking for potential overfitting that may happen when fitting the trees for classification.

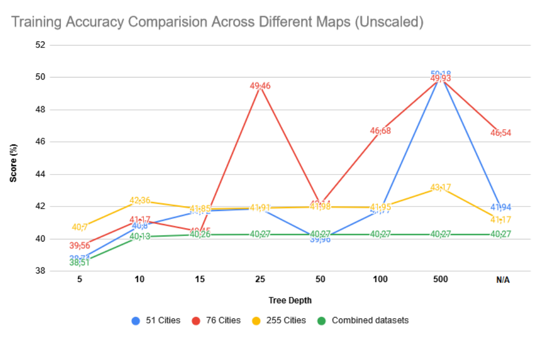
Map Instances	**Max. Tree Depth**	**Most Relevant Feature**	**2nd Most Relevant Feature**	**3rd Most Relevant Feature**
eil51	2	Avg. Fit	Avg. Var	Var. T-4
	5	Avg. Fit	Avg. Var	Var. T-1
	10	Avg. Fit	Avg. Var	Var. T-1
	15	Avg. Fit	Avg. Var	Var. T-1
	25	Avg. Fit	Avg. Var	Var. T-1
	50	Avg. Fit	Avg. Var	Var. T-1
	100	Avg. Fit	Avg. Var	Var. T-1
	500	Avg. Fit	Avg. Var	Var. T-1
	None	Avg. Fit	Avg. Var	Var. T-1
eil76	2	Avg. Fit	Var. T-4	Var. T-3
	5	Avg. Fit	Avg. Var	Var. T-2
	10	Avg. Fit	Var. T-3	Avg. Var
	15	Avg. Fit	Avg. Var	Var. T-2
	25	Avg. Fit	Var. T-2	Var. T-1
	50	Avg. Fit	Var. T-1	Var. T-2
	100	Avg. Fit	Var T-1	Avg. Var
	500	Avg. Fit	Var T-1	Avg. Var
	None	Avg. Fit	Avg. Var	Var. T-1
ts225	2	Avg. Dist	Avg. Fit	Var T-4
	5	Avg. Dist	Avg. Fit	Acorr T-1
	10	Avg. Dist	Var T-1	Avg. Fit
	15	Avg. Dist	Avg. Fit	Avg. Var
	25	Avg. Dist	Var T-1	Avg. Fit
	50	Avg. Fit	Var T-1	Var T-3
	100	Avg. Fit	Var T-1	Avg. Var
	500	Avg. Fit	Var T-1	Avg. Var
	None	Avg. Fit	Var T-1	Avg. Var
all	2	Avg. Acorr	Var T-4	Var T-3
	5	Avg. Acorr	Dist T-1	Dist T-3
	10	Avg. Acorr	Dist T-2	Dist T-3
	15	Avg. Acorr	Dist T-2	Dist T-3
	25	Avg. Acorr	Dist T-3	Dist T-1
	50	Avg. Acorr	Dist T-1	Dist T-4
	100	Avg. Acorr	Dist T-1	Dist T-4
	500	Avg. Acorr	Dist T-1	Dist T-4
	None	Avg. Acorr	Dist T-1	Dist T-4

Table 4.2: Results for the decision tree's top three most relevant important features, on each map and on the combined, standardized, dataset.

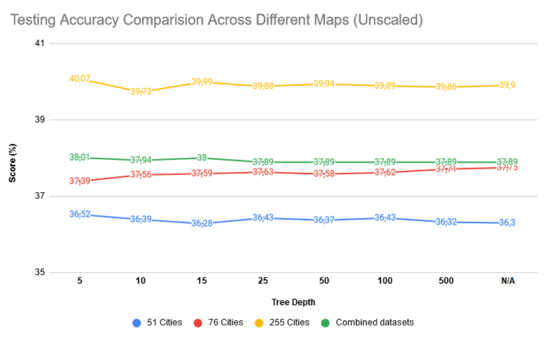
Random Tree Analysis

For this section, we opt on random forests instead of single decision tree classifiers. Similarly to above, we test different maximum depths for the forests, measuring both training and testing accuracy. Standard training/testing split is applied (70% of examples are used in training, and 30% of examples are used for testing). About 150 000 examples are used on each map, which makes for about 105 000 labeled instances and 45 000 unlabeled instances for testing. All operator classes are represented equally, with 50 000 of each being present on the evaluated set. We once again do this for all three maps, separately and combined. The random forests contain 100 estimators, and a grid search is done at every run to determine the best minimum samples required to be on a leaf node, and the minimum number of samples required for a split to happen, and use the selected configuration for display.

Furthermore, we additionally wanted to verify the impact of data scaling in our work, and as such, all instances are presented, both scaled, and unscaled, for comparison. Data scaling is done in order to standardize the range of feature values within a certain range. The results can be found on Table 4.3, and on Figure 4.20.



(a) Unscaled training accuracy results for all datasets.



(b) Unscaled testing accuracy results for all datasets.

Figure 4.20: Comparison of training/test accuracy across different maximum tree depths on unscaled datasets.

From the observation of the both the table and the figures, we can draw several conclusions: It is immediately apparent that *eil51* and *eil76* maps are more prone to overfitting, as training accuracy increases greatly with tree depth on both map instances, as is evidenced by Fig. 4.20a. There is an observable discrepancy between the training accuracy score and testing accuracy, which is a clear symptom of overfitting.

When it comes to the importance of data scaling, the results are very similar across all tests made, and we can conclude that scaling the dataset does not meaningfully affect the configurations in an impactful way.

We can further understand the classifier's behavior by looking at the normalized confusion matrices of the results, presented on Figure 4.21. While 51 and 76 city maps show a higher percentage of true positives on invert and insert operator labeling (classes 1 and 2, respectively), the 225 city map presents slightly higher accuracy, due to showing a bias toward predicting a label of swap operator (class 0) most of the time. The classifier for all the maps combined displays behavior similar to 51 and 76 city maps, with the a slight decrease on wrong predictions on swap operators, but showing a bigger spread of guesses when trying to predict an insert or invert operator.

				Unscaled		Scaled	
Map	**Max. Tree Depth**	**Min. Sample Leaf**	**Min. Sample Split**	**Train Acc. (%)**	**Test Acc. (%)**	**Train Acc. (%)**	**Test Acc. (%)**
eil51	***5***	100	100	38,74	36,52	38,95	36,16
	10	500	50	40,8	36,39	40,69	36,28
	15	500	500	41,72	36,28	41,7	36,3
	25	500	500	41,87	36,43	41,83	36,15
	50	1000	5	39,96	36,37	39,81	36,2
	100	500	25	41,77	36,43	41,73	35,92
	500	1000	10	50,18	36,32	50,09	35,91
	No Limit	500	5	41,94	36,3	41,74	36,16
eil76	***5***	50	100	39,56	37,39	40,04	37,92
	10	500	100	41,17	37,56	41,07	38,04
	15	1000	10	40,45	37,59	40,36	37,88
	25	100	25	49,46	37,63	49,35	37,67
	50	500	2	42,14	37,58	42,11	37,87
	100	100	500	46,68	37,62	46,61	37,81
	500	100	100	49,93	37,71	49,62	37,72
	N/A	100	500	46,54	37,75	46,25	37,77
ts225	***5***	1000	5	40,7	40,07	40,45	39,52
	10	250	500	42,36	39,73	42,36	39,49
	15	500	25	41,85	39,99	41,8	39,55
	25	500	250	41,91	39,88	41,74	39,65
	50	500	500	41,98	39,94	41,79	39,58
	100	500	500	41,95	39,89	41,84	39,55
	500	250	25	43,17	39,86	42,93	39,6
	N/A	1000	10	41,17	39,9	40,92	39,52
All Maps	***5***	500	25	38,51	38,01	38,91	38,14
	10	500	250	40,13	37,94	40,72	38,03
	15	1000	25	40,26	38	40,49	38,17
	25	1000	25	40,27	37,89	40,52	38,09
	50	1000	100	40,27	37,89	40,52	38,09
	100	1000	5	40,27	37,89	40,52	38,09
	500	1000	500	40,27	37,89	40,52	38,09
	N/A	1000	100	40,27	37,89	40,52	38,09

Table 4.3: Resulting training and testing accuracy scores for the random forests, alongside the parameters with best results on each map and tree depth.

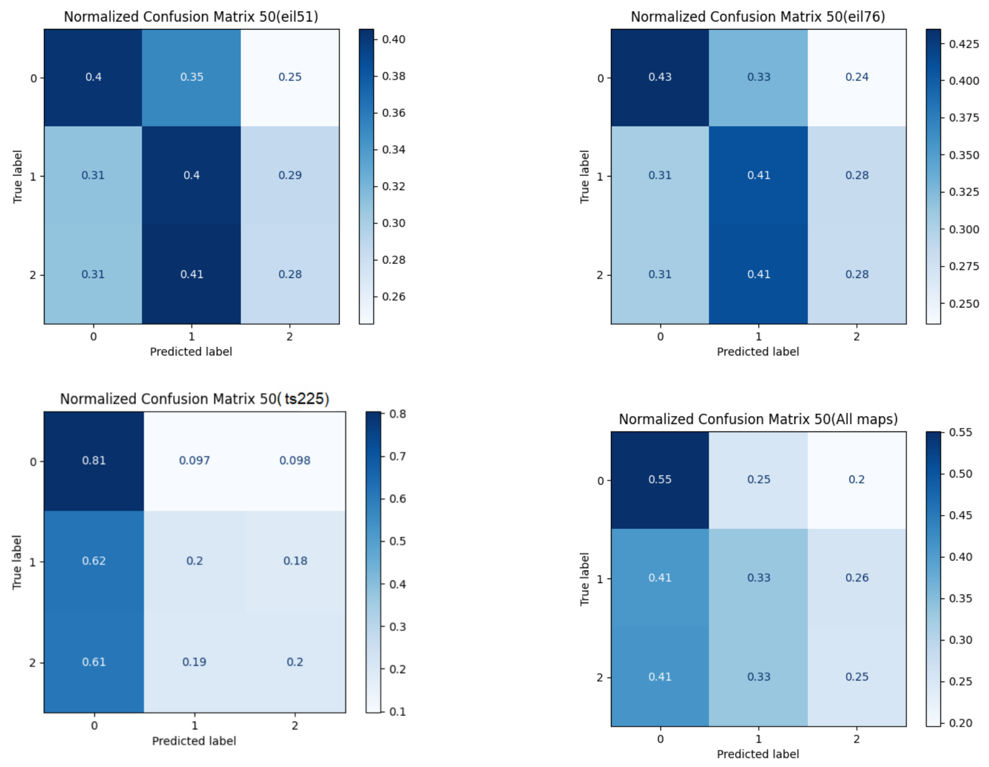


Figure 4.21: Different normalized confusion matrices obtained from the testing results for each of the map instances, with maximum tree depth of 50 and unscaled data. Swap operator is labeled 0, insert operator is labeled 1 and invert operator has label 2.

With that said, this analysis further evidences that invert and insert operators display similar behavior with one another, as is made clear by the near even distribution of incorrect labeling on both classes across all matrices displayed. Swap operator is used more often across all tests as well, evidencing that it is the operator that most promotes change in a given permutation. The low scores obtained when analyzing the results lead us to conclude that the metrics gathered throughout the work, coupled with the behavior of chosen operators makes for an unreliable model, suggesting features that further discriminate the operators is required.

Algorithm	***Average Fitness***	***Average Swap Op. Usage (%)***	***Average Insert Op. Usage (%)***	***Average Invert Op. Usage (%)***	***Last Avg. Improv. Step***
Regular HC	144438.59	33.32	33.34	33.34	149907.4
HC w/ ML Model	202571.12	6.68	92.99	0	149893.9

Table 4.4: Comparison of the average results obtained after 10 runs, including average lowest fitness value, percentage of operator selection, and average last improvement step, between a regular hill climber and hill climber enhanced with our solution, tested on the *ts225* map.

4.3 Hill Climber Comparison

Finally, in this section we compare our classifier to a hill climber algorithm. To this end we implement a hill climbing algorithm that is initialized with a random tour. At each iteration, one of the three available operators is applied randomly, and if it improves the quality of the tour, the new solution is kept, and the process is repeated iteratively until the algorithm reaches the maximum allotted iterations. For comparison, we create another hill climber separately that, in addition to measuring the fitness quality of the solution, also includes the fitness landscape metrics studied during the course of this paper. At each iteration, instead of applying a random operator, we predict which operator to use with our ML classifier, a Random Forest classifier trained on all map instances (unscaled), due to displaying a decent generalization capacity in operator selection as per Fig.4.21, even if the *ts225* classifier obtains better scores overall by applying swap operations almost exclusively.

We then display the obtained results in Table 4.4 and Figure 4.22. Note that the results are an average of 10 runs on both the simple hill climber and the hill climber enhanced with our ML algorithm, across 150 000 iterations.

As evidenced by the obtained results, and expected from the low testing scores observed in the previous sections, our ML model achieves results similar to randomly selected operators, even achieving slightly worse results. The model chooses to use the invert operator a vast majority of times, unlike what is suggested in Figure 4.21. Both of these facts combine to further suggests that the operators do not behave in a distinctive enough way for our model to be able to correctly identify when to use them, and more work needs to be done before we can properly generalize operator selection in a satisfactory manner.

As a closing remark, we can thus conclude that the model built has insufficient information to be able to make accurate predictions. This is a result of operator behavior not showing distinct enough behavior on the studied maps coupled with the obtained fitness landscape metrics, which leads us to conclude that while the observed metrics are informative in the gauging of a solution's quality, the operator to be chosen is hard to discern based on the landscape analysis alone.

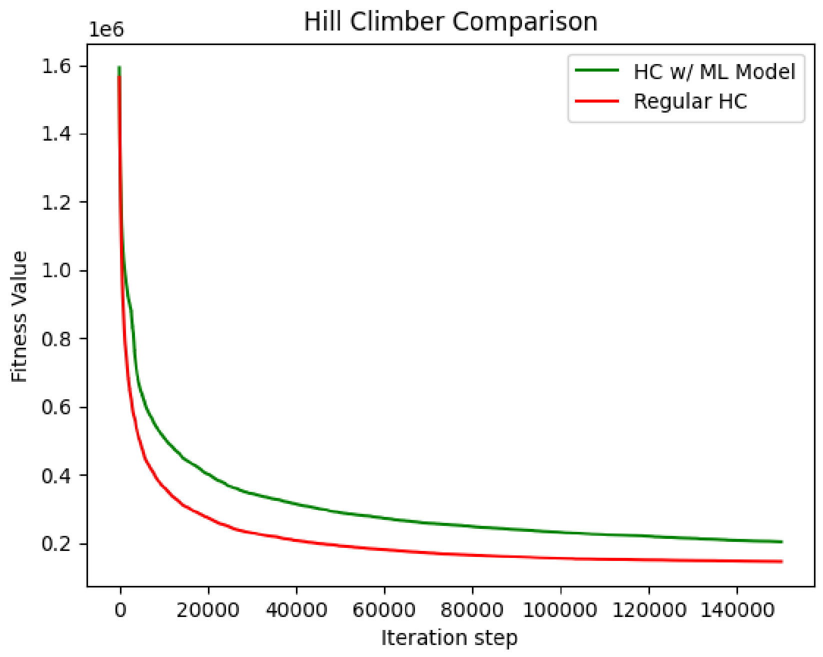


Figure 4.22: Comparison of the progression of average fitness results across 10 runs between a regular hill climber (red), and a hill climber with our incorporated ML model for operator selection (green), across 150 000 iterations, on the *ts225* problem.

This page is intentionally left blank.

Chapter 5

Conclusion and Future Work

CO problems are a challenging area studied for decades by a number of fields. In particular, the widespread rise of ML solutions in modern years has given a fresh perspective on the tackling of these problems, but as a trade-off come with a set of challenges specific to the area that need to be addressed.

Within this context, local optimization algorithms see some use, but often fall in local optima due to the complex nature of the search space found within CO problems and a number of decisions when it comes to the configuration of said algorithms are often empirically made.

In this dissertation, we explored the proposal of a framework that utilizes a ML model that takes into consideration the proprieties of the local fitness landscape of an optimization problem, and attempts to predict the best operator to be used based on that information, in order to aid an optimization algorithm in the search of an optimum.

To this end, we created a dataset composed of different instances that have metrics detailing local information on the landscape of select TSP problems. We then analyzed the behavior of the generated instances, and designed a ML model that uses Random Forests to predict operator usage, and then incorporate our model in a hill climbing algorithm and analyse the results.

Correct operator selection proved to be a complex task, as the ML displayed low capacity to correctly predict the most promising operator to be used at a given step. Furthermore, the behavior of the hill climbing algorithm shows that our solution displays results very akin to a random operator selector. This suggests a limitation not only on the metrics themselves but also on the operators used, as their behavior is not distinctive enough to be able to make accurate predictions given the information obtained from the analysis of the landscape structure.

However, we believe this work's contributions can be valuable to the evaluation of the behavior of fitness landscape metrics on their role in identifying solution quality, and in the exploration of patterns observed in the used operators' behavior, and can be further expanded in a number of different ways, as it provides a solid foundation for work in this topic.

5.1 Future Work

Our ML model was limited to the exploration of small TSP problems, and further research can be done by expanding the scope of the work. Data that better distinguishes which operator to use is crucial to our framework and as such one possibility is is exploring a bigger number of TSP maps with different sets of properties.

Additionally, the random walk we performed was relatively shallow, as it explores the local landscape with a limited set of operators. As such, potentially useful information could be gathered by applying the concept presented in our work further, by, for example, having each generate instance perform a longer walk, in conjunction with a different and/or expanded set of operators, which could potentially improve results.

Finally, the work done here was done with generalization in mind, and as such, there is opportunity for research by eventually exploring this framework in the context of different CO problems, and evaluate how it fares on situations other than the TSP.

References

[1] Mp-testdata-tsplib, 1997. URL `http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html`.

[2] E. Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[3] M. H. Alsuwaiyel. *Algorithms: design techniques and analysis*. World Scientific, 2016.

[4] D. Applegate, W. Cook, and A. Rohe. Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003.

[5] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Concorde tsp solver, 2006.

[6] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[7] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[8] J. J. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on computing*, 4(4):387–411, 1992.

[9] R. G. Bland and D. F. Shallcross. Large travelling salesman problems arising from experiments in x-ray crystallography: a preliminary report on computation. *Operations Research Letters*, 8(3):125–128, 1989.

[10] J. Caldwell, R. A. Watson, C. Thies, and J. D. Knowles. Deep optimisation: Solving combinatorial optimisation problems using deep neural networks. *arXiv preprint arXiv:1811.00784*, 2018.

[11] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[12] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau. Learning heuristics for the tsp by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*, pages 170–181. Springer, 2018.

[13] A. E. Eiben, J. E. Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[14] M. T. Goodrich and R. Tamassia. *Algorithm design and applications*. Wiley Hoboken, 2015.

[15] K. Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[16] H. H. Hoos and T. Stützle. *Stochastic local search: Foundations and applications.* Elsevier, 2004.

[17] H. H. Hoos and T. Stützle. Stochastic local search algorithms: an overview. In *Springer Handbook of Computational Intelligence*, pages 1085–1105. Springer, 2015.

[18] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[19] T. Jones, S. Forrest, et al. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *ICGA*, volume 95, pages 184–192, 1995.

[20] H. Kamoun, N. G. Hall, and C. Sriskandarajah. Scheduling in robotic cells: Heuristics and cell design. *Operations Research*, 47(6):821–835, 1999.

[21] A. R. Karlin, N. Klein, and S. O. Gharan. A (slightly) improved approximation algorithm for metric tsp. *arXiv preprint arXiv:2007.01409*, 2020.

[22] P. Kerschke, L. Kotthoff, J. Bossek, H. H. Hoos, and H. Trautmann. Leveraging tsp solver complementarity through machine learning. *Evolutionary computation*, 26(4): 597–620, 2018.

[23] B. Korte, J. Vygen, B. Korte, and J. Vygen. *Combinatorial optimization*, volume 2. Springer, 2012.

[24] L. Kotthoff, P. Kerschke, H. Hoos, and H. Trautmann. Improving the state of the art in inexact tsp solving using per-instance algorithm selection. In *International Conference on Learning and Intelligent Optimization*, pages 202–217. Springer, 2015.

[25] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[26] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

[27] M. Malek, M. Guruswamy, M. Pandya, and H. Owens. Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals of Operations Research*, 21(1):59–84, 1989.

[28] C. M. Martinez, M. Heucke, F.-Y. Wang, B. Gao, and D. Cao. Driving style recognition for intelligent vehicle control and advanced driver assistance: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 19(3):666–676, 2017.

[29] P. Merz and B. Freisleben. Memetic algorithms for the traveling salesman problem. *Complex Syst*, 14, 07 2002.

[30] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning.* MIT press, 2018.

[31] Y. Nagata. Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem. In *Proc. of 7th Int. Conf. on Genetic Algorithms, 1997*, 1997.

[32] Y. Nagata and S. Kobayashi. A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *INFORMS Journal on Computing*, 25 (2):346–363, 2013.

[33] J. Pihera and N. Musliu. Application of machine learning to algorithm selection for tsp. In *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, pages 47–54. IEEE, 2014.

[34] Y. Rossier, M. Troyon, and T. M. Liebling. Probabilistic exchange algorithms and euclidean traveling salesman problems. *Operations-Research-Spektrum*, 8(3):151–164, 1986.

[35] S. Russell and P. Norvig. Artificial intelligence: a modern approach. 2002.

[36] A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

[37] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[38] S. S. Skiena. *The algorithm design manual: Text*, volume 1. Springer Science & Business Media, 1998.

[39] P. F. Stadler and W. Schnabl. The landscape of the traveling salesman problem. *Physics Letters A*, 161(4):337–344, 1992.

[40] G. Tao and Z. Michalewicz. Inver-over operator for the tsp. In *International Conference on Parallel Problem Solving from Nature*, pages 803–812. Springer, 1998.

[41] J. Tavares, F. B. Pereira, and E. Costa. Multidimensional knapsack problem: A fitness landscape analysis. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(3):604–616, 2008.

[42] M.-H. Tayarani-N and A. Prügel-Bennett. An analysis of the fitness landscape of travelling salesman problem. *Evolutionary computation*, 24(2):347–384, 2016.

[43] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[44] L. WANG, A. Maciejewski, H. SIEGEL, V. Roychowdhury, and B. ELDRIDGE. A study of five parallel approaches to a genetic algorithm for the traveling salesman problem. *Intelligent Automation & Soft Computing*, 11, 01 2005. doi: 10.1080/10798587.2005.10642906.

[45] E. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological cybernetics*, 63(5):325–336, 1990.

[46] S. Wright. *The roles of mutation, inbreeding, crossbreeding, and selection in evolution*, volume 1. na, 1932.

[47] X.-s. Yan, H.-m. Liu, J. Yan, and Q.-h. Wu. A fast evolutionary algorithm for traveling salesman problem. In *Third International Conference on Natural Computation (ICNC 2007)*, volume 4, pages 85–90. IEEE, 2007.