Utrecht University

Department of Information and Computing Sciences

# Investigation of the Traveling Thief Problem

*Author:*
Rogier Hans Wuijts
(ICA-4001184)

*Supervisor:*
dr. ir. D. Thierens

June 11, 2018

**Abstract**

Traveling Thief Problem (TTP) is a relatively new benchmark problem created to study problems which consist of interdependent subproblems. In this thesis various operators and strategies in the literature of TTP are investigated in order to understand if and how they work and whether they can be improved. Operators include 2-opt, Insertion, Bitflip and Exchange. Strategies include greedy packing plan heuristic, neighborhood reduction and use of various tour crossovers in a genetic algorithm. The investigation is done in part by fitness landscape analysis. The first part of the thesis is a literature study and discusses TTP, the subproblems of TTP and fitness landscape analysis. The second part of the thesis consists of experiments which investigate these various operators and strategies in the literature. The second part also consists of time complexity improvements for the commonly used local search operators for TTP. Experiments show that greedy packing heuristics have a high chance of finding optimal or near-optimal solutions. Other experiments show that between nearest neighbor reduction, k-quadrant nearest neighbor reduction and reduction by Delaunay triangulation there is no significant difference in performance. Fitness landscape analysis shows among other things that TTP instances contain a lot of local optima but their distance to the global optimum is correlated with its fitness. Local optima networks with respect to an iterated local search shows that TTP has a multi-funnel structure. Other experiments show that a steady state genetic algorithm with edge assembly crossover outperforms multistart local search, iterated local search and genetic algorithms with other tour crossovers. At last there is a comprehensive comparative study which contains new best solutions to almost all studied instances of the commonly used benchmark suite.

# Contents

# 1 Introduction

The Traveling Thief Problem (TTP) was created in 2013 by Bonyadi, Michalewicz, and Barone [7]. In TTP a thief must visit each city once and at each city has the opportunity to take items with him. These items have a certain profit and weight and the knapsack has a certain capacity, the thief can't take all items.

The TTP is a combination of two NP-hard subproblems: Knapsack Problem (KP) and the Traveling Salesman Problem (TSP). The two subproblems are interdependent because the weight of the items slows the thief down and time is of importance since the knapsack is rented. The speed of the thief has a non-linear relation with the weight the thief carries. The goal of the traveling thief is to maximize the profit from items while paying rent for the knapsack.

The problem may sound a little bit goofy but interpretation aside the combination of KP and TSP with an interdependence is an interesting problem. The optimal knapsack depends on the route the thief takes and the optimal tour depends on the items in the knapsack. Finding the optimal knapsack and tour is harder than solving each subproblem individually. That is the crux of TTP.

TTP was created because Bonyadi, Michalewicz, and Barone found that research about and comparison between metaheuristics takes place in the context of certain NP-hard problems. But they argue that there is a growing gap between real problems and these benchmark problems. The growing gap comes from the fact that the problems encountered in the world become increasingly more complex while the benchmark problems in 50 years have stayed the same. The traveling thief problem tries to close this gap by being a problem with two important characteristics of real-world problems: It contains two subproblems and there exists an interdependence between the two.

In the short time since TTP has been introduced various algorithms have been proposed. But as I will explain in section 4.8.1 there is no real justification for using these operators/strategies besides some comparison between different algorithms or previous findings from TSP literature. Therefore I would like to research two things in my thesis: investigate and understand these operators/strategies and try to improve them. I will investigate the previous point in part by fitness landscape analysis in order to answer the following research question:

**How can the use and effect of various operators and strategies in the literature of the traveling thief problem be justified, explained and improved?**

The structure of this thesis is as follows: part I (section 1-6) is a literature study of TTP, TSP, KP and fitness landscape analysis. Part I also contains explanations of various local search procedures, metaheuristics and operators used in this thesis. In section 6 I will give a detailed explanation of my research question and define subquestions which I will answer in part II. In part II of my thesis I will answer these subquestions at each section and in section 14 give a conclusion.

## 1.1 Definition

Traveling thief problem formally defined:

Given $n$ cities, $m$ items, $m_i$ items at city $i$, a profit $p_{ik}$ and weight $w_{ik}$ for each item $k$ at city $i$, a distance $d_{ij}$ between city $i$ and $j$, a minimum speed $v_{min}$, a maximum speed $v_{max}$, a renting rate $R$ and a knapsack capacity $C$. The goal is to find a permutation of cities $\Pi = (x_1, ..., x_n)$ and a bitstring representing the packing plan $Y = (y_{21}, ..., y_{2m_2}, ..., y_{nm_n})$ such that the objective $Z(\Pi, Y)$ is maximized. $Z(\Pi, Y)$ is defined the following way [56]:

$$Z(\Pi, Y) = \sum_{i=1}^{n} \sum_{k=1}^{m_i} p_{ik} y_{ik} - R \sum_{i=1}^{n} \frac{d_{x_i x_{s(i)}}}{V_i} \tag{1}$$

Where $s(i)$ is the successor of integer $i$ in the permutation $\Pi$. $Z$ is the summation of profit of the packed items minus the renting rate times the travel time. The travel time depends on the distance between city $x_i$ and its successor and the speed $V_i$.

$$V_i = v_{max} - \frac{v_{max} - v_{min}}{C} W_i \tag{2}$$

The speed $V_i$ at city $x_i$ is calculated with the collected weight $W_{x_i}$ from items at city $x_i$. Without any items the speed is equal to $v_{max}$ and with full capacity it is equal to $v_{min}$.

$$W_i = \sum_{j=2}^{i} \sum_{k=1}^{m_j} y_{x_{jk}} \cdot w_{x_{jk}} \tag{3}$$

$$W_{x_n} \leq C \tag{4}$$

The collected weight at the last city $x_n$ must be smaller or equal to capacity $C$.

## 1.2 Outline Part I

The structure is as follows: In section 2 I will introduce local search that is relevant for TTP. In section 3 I will give an overview of the literature of KP and TSP. In section 4 there will give a complete overview of the TTP literature. In section 5 I will introduce a subset of landscape analyses I want to use in my thesis and at last in section 6 I will propose my research questions with relevant subquestions.

# Part I
# Literature Study

## 2 Preliminary

### 2.1 Local Search

A local search is a method of iteratively improving a solution by making local changes. These local changes are made with an operator. Most of the time the operator can be applied in multiple ways. Every application of an operator creates a neighbor solution. Iteratively going to a better neighbor solution is called a hill climbing algorithm (see Algorithm 1). If a solution has no neighbors with a better evaluation score, then the solution is a local optimum with respect to the operator. For the traveling thief problem, local searches are performed on the packing plan or on the tour. The packing plan is represented by a bit string and the tour by a permutation.

In the next section I will briefly explain some local search procedures that are used throughout this thesis. When a local search is mentioned it is implied this procedure is done until no improvements are made or stated otherwise. Sometimes a local search is only applied for one pass and sometimes an operator is randomly applied. In the latter case it serves as a disruptive operator.

**Algorithm 1** General format hill climbing (best improvement)

```
1: function HILL CLIMBING
2:     x ← an initial solution
3:     while termination conditions are not met do
4:         x ← arg max{f(x') | x' ∈ N(x)}        ▷ f is a fitness function
5:                                                ▷ N is the neighborhood of x
6:     end while
7: end function
```

#### 2.1.1 2-opt

2-OPT is a local search procedure created for TSP but can be applied to any kind of permutation representation. The 2-OPT-swap reverses a segment of the permutation. In terms of a tour it deletes and creates two edges (Figure 1).

Since every segment can be used for a valid application of 2-OPT-swap, this operator has a total $O(n^2)$ neighboring solutions.

#### 2.1.2 Insertion

Closely related to 2-OPT is INSERTION also called 2.5-OPT. Here an element of the permutation is chosen and reinserted somewhere within the permutation.

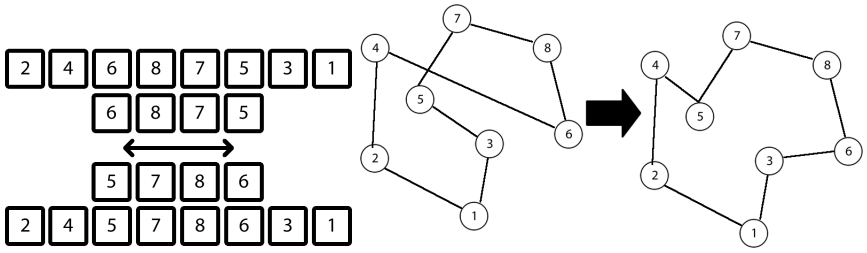


Figure 1: 2-OPT. Permutation representation (left) and the corresponding tour (right).

There are $O(n)$ candidates for insertion and $O(n)$ places to insert, again $O(n^2)$ neighboring solutions.

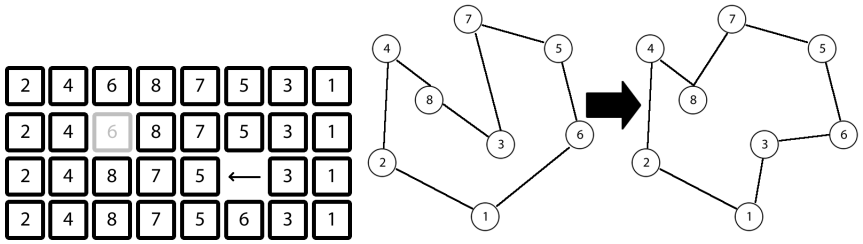


Figure 2: INSERTION. Permutation representation (left) and a corresponding tour (right).

### 2.1.3 BitFlip

As the name probably suggests BITFLIP is a local search that flips a bit of a bit string. In terms of the packing plan this is either inserting or removing an item from the knapsack. There are $O(n)$ possible bits to flip.

### 2.1.4 Exchange

EXCHANGE is a local search that takes two bits with different values and swaps their status. In terms of the packing plan this is consecutively removing and inserting an item. There are $O(n^2)$ possible combinations. This local search procedure is stronger than BITFLIP since it can first 'visit' an invalid or worse solution by first removing or inserting an item.

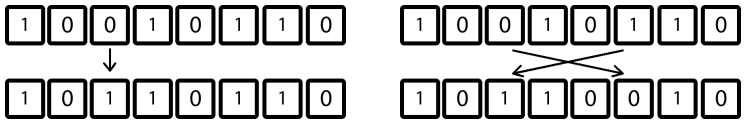


Figure 3: BitFlip (left) and Exchange (right)

# 3 Subproblems of the Traveling Thief Problem

In this section I want to provide a small literature study on the traveling salesman problem and the knapsack problem to gain insight into the subproblems of the traveling thief problem. Not by any stretch of the imagination will this be a complete overview of the work done in these fields. I have chosen parts of the literature which I deem historically relevant, relevant to the TTP and the TTP-literature or relevant to paths I take in my thesis.

## 3.1 Traveling Salesman Problem

The traveling salesman problem is one of the most intensely investigated problems in optimization [1]. Its popularity is due to its easy to understand but hard to solve nature. It also has a large variety of applications in many different areas. To name a few: logistics, mapping genomes, aiming telescopes, guiding industrial machines, scheduling jobs and many more [10]. The pursuit of solving TSP, be it exact or approximate, has pushed research in computer science forward, created novel methods and algorithms along the way and functioned as a test-bed for various metaheuristics.

The traveling salesman problem can be defined as finding the shortest Hamiltonian cycle in a graph. The problem is finding a permutation of cities such that the cost of the summation of every edge between consecutive cities is minimized. The search space of this problem is enormous: $(n-1)!$ different solutions exist for asymmetric TSP and $\frac{(n-1)!}{2}$ for symmetric TSP. A brute force algorithm would suffer from the giant permutation search space and is therefore not feasible.

### 3.1.1 Exact Methods

A significant but still impractical improvement has been made by the Held–Karp algorithm that solves TSP with dynamic programming and has an improved time bound of $O(n^2 2^n)$ [21]. This still takes too much time for even the smallest instances.

The state-of-the-art of using an exact methods to solve TSP is linear programming using branch-and-cut. The most efficient implementation of this method is in the freely available Concorde [19]. Concorde is software that contains approximately 130.000 lines of code and lies at the heart of the research done by Applegate et al. [1]. It includes multiple methods that have been accumulated over more than 60 years of LP-reseach into TSP. Concorde can solve instances up to 85900 nodes, for an example the instance $pla85900$ of TSPLIB [57]. The computation however comes at the cost of consuming over 136 CPU years on a cluster of computers. Instances of smaller size, i.e. 1000 nodes, only takes minutes. Linear programming is the most successful exact TSP approach proposed to date and Applegate et al. called it one of the great success stories of modern mathematics.

### 3.1.2 Construction Algorithms

Given the hardness of the problem a lot of research on TSP went into heuristic algorithms. One of the earliest approaches is to simply construct a tour following a basic rule. In the nearest neighbour algorithm a tour is created by iteratively going to the nearest city. This naive algorithm performs reasonable well for the first few cities but will eventually require long edges to go back to unvisited cities. In fact Gutin, Yeo, and Zverovich [18] showed that it can return the worst possible tour.

Other construction algorithms have been proposed which give a guaranty on the quality of the solution: approximation algorithms. One of the best approximation algorithms to date is the Christofides algorithm [9]. The Christofides algorithm finds a tour in the following way: First, a minimal spanning tree $T$ is created. Then a minimum-weight perfect matching $M$ in the vertices of odd degree in $T$ is found. After which an Euler tour is constructed in $T \cup M$. Removing duplicate vertices in this Euler tour gives a TSP-tour with a quality that is no less than $\frac{3}{2}$ of the optimal solution. In practice the tour is much closer than $\frac{3}{2}$ of the optimum but it is still outperformed by a simple hill climber algorithm using 2-opt [23].

### 3.1.3 Lin–Kernighan Heuristic

More practical success has been achieved by various applications of the Lin–Kernighan heuristic developed by Lin and Kernighan [28]. The Lin–Kernighan heuristic (LK) is generalization of the 2-opt search and is the backbone of many metaheuristics that solving TSP [1]. LK iteratively improves the tour by searching for a sequence of edge swaps such that each initial subsequence has a chance of improving the tour. If it actually does improve the tour, the sequence of edge swaps is executed. If no sequences can be found, a local optimum has been reached and the algorithm stops.

To find a better solution you could run LK multiple times (MLS) or perturb the solution and apply the local search again (ILS). The Chained Lin-Kernighan heuristic does exactly that. It kicks a solution out the basin of attraction of its local optimum by a non-sequential move called the double bridge. The double bridge is an exchange of 4 edges such that a Lin-Kernighan search cannot find the set of flips needed to undo the exchanged edges. To date the Chained Lin-Kernighan heuristic is still one of the best approaches for very large data sets ($n > 25.000.000$) [10].

One rather effective implementation of this LK heuristic is the one by Helsgaun, abbreviated as LKH [22]. LKH has various optimizations that speed up the algorithm as well as some significant changes that alter the structure of the algorithm. Instead of using the 2-opt exchange as base move it uses a 5-opt exchange, considering 10 edges at the same time. One other change is that LKH only considers promising edges to swap. Promising edges are those that are likely to be found in the optimal solution. It finds these edges by a measure called $\alpha$-measure which is defined as the distance to the corresponding minimal

1-tree, a spanning tree with one additional edge. A minimal 1-tree has been empirically proven to contain between 70-80% of the edges of an optimal tour [22]. LKH held the record for multiple TSP instances and already found the global optimum of *pla*85900 before Applegate et al. did but in considerable less time [1].

## 3.2 Genetic Algorithms and Permutation Crossover

Although early genetic algorithms for the TSP were not successful [10], relatively new genetic algorithms with permutation crossovers like EAX and GPX give comparative results with Chained Lin-Kernighan heuristic and LKH [44, 58].

An important aspect of a genetic algorithm is its recombination operator: its crossover. Finding a good crossover for permutations is not trivial. Using a traditional crossover like 1-point or uniform crossover is not an option (Figure 3.2).

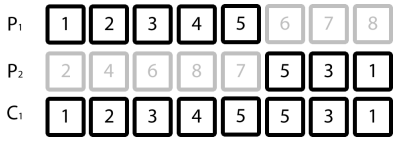


Figure 4: Example of where 1-point crossover results in an invalid solution

In the last 30 years numerous crossover operators were defined for the permutation representation of the TSP. Different crossovers focus on different aspects of the problem. Some value absolute position, like cycle crossover (CX), position crossover (PX), partially-mapped crossover (PMX) and to some extent order crossover (OX). Other crossovers value relative order, like maximal preservative crossover (MPX) and OX. The last group values adjacency information, like edge recombination crossover (ERX), partition crossover (PX), generalized partition crossover (GPX) and edges assembly crossover (EAX). Adjacency information and relative order are important information for TSP but absolute position is not [27]. In general for the TSP problem it holds that $CX < PMX < OX < ERX$ [49, 60].

In this section I would like to discuss crossovers that have been proposed for TSP. First some that are outperformed by other methods but are still relevant since they might be useful for TTP and occur in the TTP literature. At last I will introduce some that are effective and can compete with the state-of-the-art heuristics for TSP.

### 3.2.1 Cycle Crossover (CX)

The cycle crossover was proposed by Oliver, Smith, and Holland [49]. It works the following way: First cycles are constructed for both parents. To create a cycle, start at a random city $c$ of the first parent $P_1$ that is currently not in

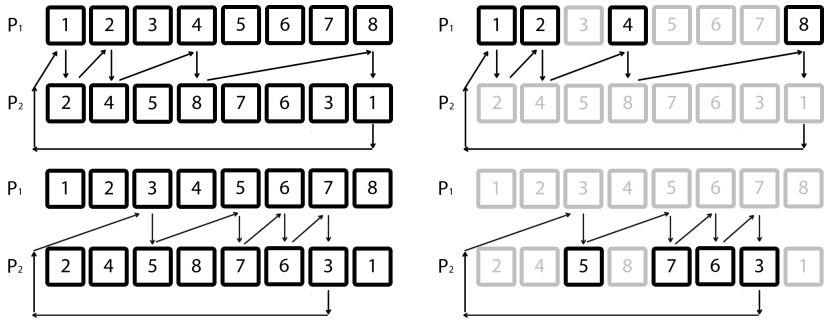


Figure 5: Example of two cycles (left) and a cycle crossover (right)

any cycle. The next city will be chosen to be at the position of $c$ at the second parent $P_2$. Repeat this until you encounter the initial city, creating a cycle.

The offspring only inherits all or none of the nodes of a cycle belonging to the same parent. The city at every position of the offspring of CX corresponds to either $P_1$ or $P_2$. CX correspond to doing uniform crossover on the cycles.

The cycle crossover preserves the absolute position of the two parents, relative order only within a cycle and adjacency information only by accident.

### 3.2.2 Order Crossover (OX)

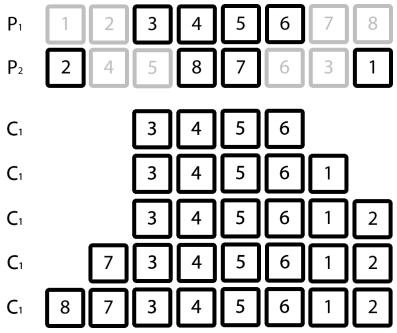


Figure 6: Example of a order crossover

The order crossover was first introduced by Davis [12]. It works the following way: first a segment is copied from one parent to the offspring. This segment is defined as the cities between two randomly determined crossover points. Then the remaining cities are added after the second crossover point. They are added in the order they appear in the other parent starting from the second crossover point.

The order crossover preservers the absolute position, relative order and adjacency information of the copied segment but only the relative order of the other elements.

### 3.2.3 Partially-Mapped Crossover (PMX)

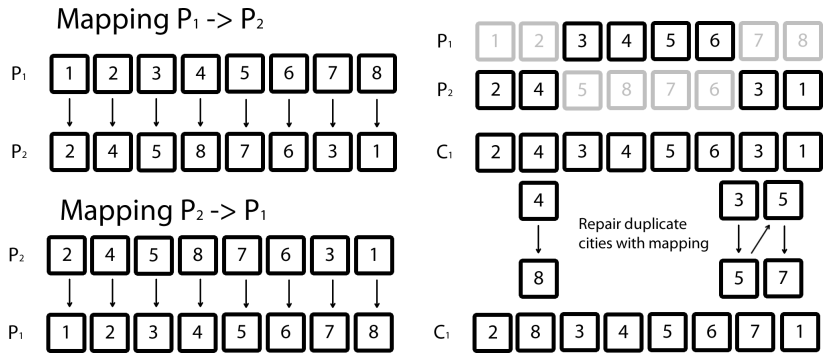


Figure 7: Example of a mapping between parent (left) and a partially-mapped crossover (right)

Partially-mapped crossover was introduced by Goldberg, Lingle, et al. [17]. It works the following way: First a segment is copied from one parent (same as in OX). Then the remaining positions are copied from the other parent. This results in duplicate cities. In order to repair this the mapping between nodes of the parents is used. If the city is still duplicate the mapping is applied again until the city is unique in tour.

### 3.2.4 Maximal Preservative Crossover (MPX)

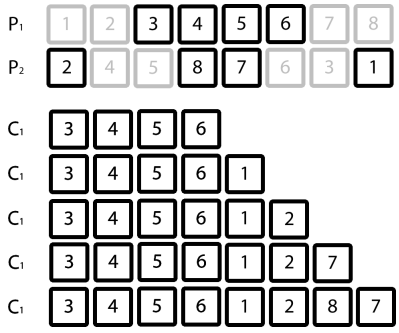


Figure 8: Example of a maximal preservative crossover

The maximal preservative crossover was introduced by Muhlenbein [42] and works the following way: first a segment is copied from one parent to the front of the tour. After the segment is copied, cities are added from the other parent in the order they appear.

The segment was originally defined as the cities between two randomly determined crossover points with a restriction that the segment is greater than 10

but smaller than $\frac{n}{2}$ [27]. Another study found that using a fixed length of $\frac{1}{3}$ gave better results [34]. The relative position and adjacency information of one parent is preserved while only the relative order of the other parent is inherited. MPX has good results compared to other classical permutation crossovers in combination with a local search [67].

### 3.2.5 Crossover with random keys

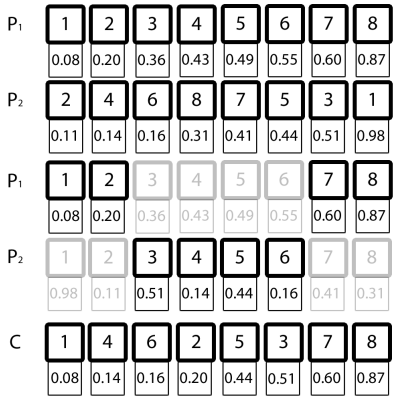


Figure 9: Example of a crossover with random keys

Another way to do crossover on two permutations is by using a random keys encoding instead of a direct path encoding. Representing the permutation as a list of random keys was first proposed by Bean [4]. First a list on random numbers is created then the list is sorted and every city is assigned to a number on the list corresponding to the relative order of the permutation. Applying traditional crossover operators on the random key encoding will always result in a valid permutation. In Figure 9 I have used a two-point crossover.

### 3.2.6 Edge Recombination Crossover (ERX)

The genetic edges recombination crossover was introduced by Whitley and Starkweather [69] and works in the following way: First an edge map is created containing adjecency information of parents. Then the current city $c$ is chosen at random. All occurrences of $c$ are removed from the edge map. The next value of $c$ is chosen from the available adjacent city. If there is no adjacent city then a random unvisited city will be picked.

When picking the next city, the ones with the fewest entries gets priority to ensure cities do not become isolated. In general we want to introduce the least number of new edges. Starkweather et al. [60] made an enhancement to ERX by also prioritizing edges that are present in both parents.

The ERX preserves the adjacency information of both parents but the relative order can be disrupted since ERX can reverse the direction.

### 3.2.7 Edge-Assembly Crossover (EAX)

A really effective crossover, the edge-assembly crossover, was proposed by Nagata [43]. The procedure involves finding AB-cycles. AB-cycles are constructed from the two parents, which Nagata names Tour-A and Tour-B. An AB-cycle is a cycle consisting of alternating edges of Tour-A and Tour-B. EAX is performed the following way:

1. Construct graph $G_{ab}$ by merging tour-A and tour-B
2. Divide the edges in $G_{ab}$ into AB-cycles
3. Construct an E-set by selecting cycles according to a given rule
4. Generate an intermediate solution by transforming Tour-A. Remove edges from Tour-A which occur in the E-set and add the edges from tour-B in the E-set.
5. Connect the subtours heuristically

Figure 10 can give the much needed intuition. Selecting an E-set can either be done by selecting one cycle at random, in which case the offspring is similar to one of its parents or by selecting multiple AB-cycles and in that case the offspring inherits a roughly equal amount of edges from Tour-A and Tour-B.

EAX is one of the most successful tour-finding procedures finding. It can find good solutions even without the use of a local search procedure like LK [10].

### 3.2.8 (Generalized) Partition Crossover ((G)PX)

The partition crossover is a relatively new crossover operator introduced by Whitley, Hains, and Howe [67]. It creates offspring by only using edges present in both parents. It does this by combining the two parents and merging common edges. After which it tries to find a cut of cost two (It tries to partition $N$ in two sets, $S1$ and $S2$, such that there are only 2 edges going from $S1$ to $S2$). The offspring inherits all common edges. The PX preserves adjacency information since the offspring is created using only edges present in both parents.

When partition crossover is applied to parents that are local optima, the offspring is usually also a local optimum and in 50% of the cases of applying the crossover the offspring has an improved fitness [67]. Only inheriting edges from parents and never introducing new edges also has its downside. If edges from the global optimum are not present in the population then PX alone can never find the global optimum.

Whitley, Hains, and Howe [66] realized that when there are multiple partitions in a graph there are also multiple offspring possible. If there are $k$ partition components then there are $2^k - 2$ distinct offspring. The TSP objective function is of such a nature that you can greedily construct the best possible offspring in $O(n)$. This procedure is called the generalized partition crossover (GPX).

GPX suffers from the same problem PX has, edges are never introduced. Sanches, Whitley, and Tinós [58] tried to solve this by combining GPX which is highly exploitative with EAX which is highly explorative. Empirical results

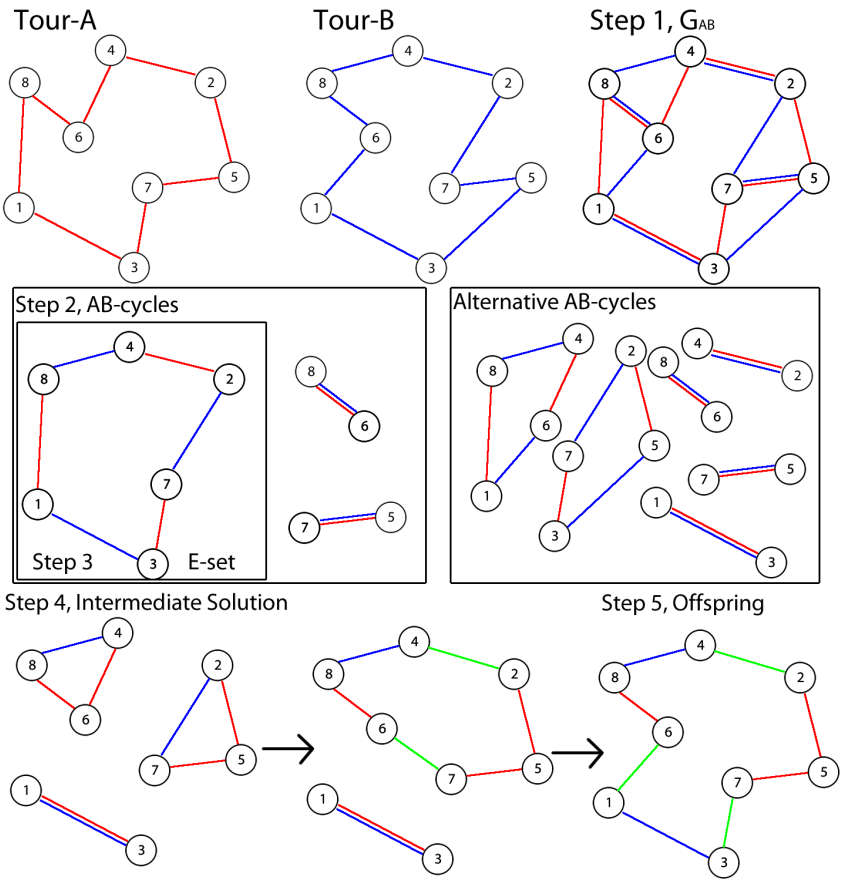


Figure 10: Example of an edge-assembly crossover. Red edges are from Tour-A, blue edges from Tour-B and green edges are foreign.

demonstrate that combining the two can lead to better results than applying only one of them.

### 3.2.9 TSP as Subproblem of TTP

As mentioned before the traveling salesman problem is part of the traveling thief problem. In fact if no items are picked up or the renting rate $R$ is set to an extremely high value then the subproblem becomes TSP.

One major difference is that not only distance between cities counts towards the objective function but that the total travel time of the whole tour is important. Travel time depends on the speed of the thief and thus on the items that are picked up. It also depends on what order they are picked up. In general it is a good idea to pick up heavy items at the end of the tour and if heavy items are pick up visit the corresponding cities at the end of the tour. Another

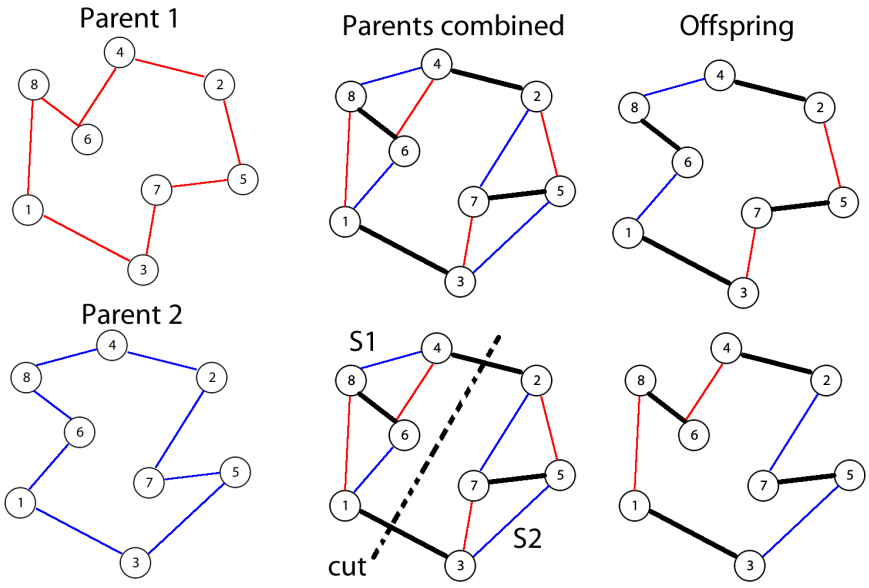


Figure 11: Example of a generalized partition crossover

difference between TSP and TTP is that there exists an initial city. Therefore while relative order and absolute position might not play a roll in TSP, this is important for TTP.

Using travel time instead of distance has another downside: small changes effect the whole tour and thus incremental fitness evaluation is not possible. This is a major downside since it multiplies the time complexity of almost all TSP heuristics by a factor $O(n)$. This also implies that algorithms, for example LK, and operators, for example EAX or GPX, aren't trivially translated to TTP. LK makes decisions based on edge length and EAX recombines partial solutions in a greedy way. The question remains if those recombination operators can be used efficiently and effectively.

## 3.3 Binary Knapsack Problem

The knapsack problem is a popular optimization problem which often occurs as a subproblem in algorithmic approaches that solve real world problems [59]. In the knapsack problem, one tries to find the set of items with the most combined profit while respecting the capacity constrain. More formally, given $m$ items with weight $w_i$ and profit $p_i$ and capacity of the knapsack $C$. Let $x_i$ be the decision variable that is 1 when the item is included and 0 otherwise. The knapsack problem can be formulated as an ILP in the following way:

$$\text{maximize} \sum_{i=1}^{m} p_i x_i$$

$$\text{subject to} \sum_{i=1}^{m} w_i x_i \leq C$$

$$x_i \in \{0, 1\}$$

The special case of knapsack where $p_i = w_i$ is called the subset sum problem. Knapsack is NP-Hard in the weak sense meaning that although there doesn't exist a polynomial algorithm to solve knapsack, there does exist a pseudo-polynomial algorithm. Most of the carefully constructed hard knapsack problems can be solved with Dynamic Programming in milliseconds [52].

Dantzig [11] introduced a procedure to get a good solution by simply sorting the items on there profit/weight ratio such that:

$$\frac{p_j}{w_j} \geq \frac{p_{i+1}}{w_{j+1}} \forall i \in \{1, ..., m-1\}$$

and greedily adding them in this order to the knapsack. The first item that cannot be included in the knapsack is called the break item. Items added to the knapsack greedily give an optimal solution to the continuous knapsack and a pretty good solution to the binary knapsack problem. The solution to the continuous knapsack can also be used as an upper bound to the binary knapsack problem.

### 3.3.1 Exact Method

Successful knapsack algorithms are usual based on two approaches, branch-and-bound and dynamic programming [32]. Most of these use the Dantzig integer solution as initial solution or use an upper bound of the just mentioned continuous solution. Tighter bounds are found by calculating the maximum cardinality. Additional speedups can be achieved by multiplying the coefficients.

One way of solving knapsack is with core algorithms that only consider interesting items called the core. The core is defined as the set of items around the break item. All items with a ratio higher than all core items are included in the knapsack and all items with a ratio smaller than all core items are excluded. If the core is chosen correctly this solves the problem to optimality and if not it can be used as an upper bound. Choosing the correct core can be difficult. Pisinger [51, 50] solves this by finding the correct core on the fly by iteratively adding items to the core or removing them from the core. The core itself can either be solved with a ILP or with dynamic programming.

## 3.4 Hardness of an Instance

Most of the knapsack instances studied in the literature are artificially generated. The hardness of a knapsack instance in part depends on the chosen correlation between the profit and the weight. An instance where the profit and weight are chosen independently are called uncorrelated and are easy to solve [32]. This comes from the fact that the upper bounds can easily identify which item won't contribute to an optimal solution. Instances where the profit highly depends on

the weight, $p_i = w_j + \epsilon$ are much harder to solve. Two observations by Pisinger [52] why they are harder: 1) There is a large gap between the continuous and integer solution of the problem. 2) For any small interval of ordered items there is a limited variation in weights making to difficult to fill up the knapsack to maximal capacity.

Using higher coefficients will also increase the computation time for dynamic programming and Martello, Pisinger, and Toth [32] suggest that instances with an exponentially growing coefficient will remain hard to solve due to the NP-hardness of the problem.

## 3.5 Knapsack as Subproblem of TTP

Knapsack is a subproblem of TTP. If the renting rate $R$ or the distances between cities are set to zero then TTP becomes knapsack.

The major difference between knapsack and knapsack in TTP is that the decision to pick up an item doesn't only depend on the weight and profit but also on the total traveling time. This makes the property of weight-profit ratio far less important than in the normal knapsack problem which is key to almost all efficient knapsack algorithms.

The gain of an item depends on the tour but also depends on the status of other items. The situation in which the tour is kept fixed is a problem on its own called Packing While Traveling (PWT) [54]. Polyakovskiy and Neumann [55] solve PWT by first using a pre-processing scheme that decreases the amount of items by directly including or discarding certain items. Then they solve the reduced instances with either one of two exact approaches, one using constraint programming with the branch-and-cut method and one using mixed-integer programming.

Neumann et al. [45] later proposed a more efficient exact approach with dynamic programming that solves PWT in pseudo-polynomial time. In the same paper they proposed a fully polynomial time approximation scheme for a variant of PWT.

Early attempts to translated these results can back to TTP where made [16] and it remains interesting to see if the dynamic programming of Neumann et al. [45] can be used as part of an algorithm to solve TTP.

# 4 Traveling Thief Problem

## 4.1 Origin

In their original paper Bonyadi, Michalewicz, and Barone [7] propose two problem variants of the traveling thief problem, $\text{TTP}_1$ and $\text{TTP}_2$. $\text{TTP}_2$ is a bi-objective optimization problem. The two objectives are: minimize the total travel time and maximize the profit with an additional feature that the value of an item drops over time. This variant received little attention in the literature.

$\text{TTP}_1$ is the variant that is studied most in the literature and is also the subject of this thesis. It is important to note that the definition of $\text{TTP}_1$ is slightly different from the commonly used TTP definition. In the original paper the thief could pick the same item at different locations while most research now focuses on the variant of Polyakovskiy et al. [56] in which items are limited to only one city. See definition in Section 1.1.

## 4.2 Difficulty of the Problem

Simply putting two subproblems together with an interdependence does not necessarily lead to an interesting or hard problem but there is ample evidence that suggests it is indeed hard to find a good solution for instances of TTP:

- Bonyadi, Michalewicz, and Barone [7] showed in the original paper that a global optimum to one of the two subproblems is not necessarily a global optimum of the whole problem. Indeed for some instances solutions with much longer tours are found to have better fitness values than those that have a tour that lies closer to TSP-optimal solutions [29, 41, 63].
- Mei, Li, and Yao [37] investigated the interdependence between the two subproblems and stated that the non-linear interdependence of the subproblems makes it difficult to decompose the problem into independent subproblems, if not impossible.
- El Yafrani and Ahiod [13] showed that it is impossible to recover the objective value of a mutated solution in a constant time. The mutations they used where 2-OPT and BITFLIP. Therefore incremental fitness evaluation is not possible and the use of local search is much more computational expensive than for example with TSP.

The fact that incremental fitness evaluation is not possible makes the problem time consuming to solve even the smallest of instances. The fitness evaluation for TTP is $O(n + m)$. When only the tour changes but the packing plan remains the same the fitness evaluation can be improved to $O(n)$ by storing the total profit and weight. When the packing plan remains the same for each city these totals do not change [36].

## 4.3 Benchmark

Polyakovskiy et al. [56] provided a benchmark suite in order to study TTP. The benchmark is build upon instances of the TSP library, TSPLIB [57]. Only TSP instances where the distance between two cities is defined as the euclidean distance are considered. A TSP instance from the TSPLIB specifies the amount of cities and the distances between them. In order to turn a TSP instance into a TTP instance a few things need to be done. Items need to be generated, items need to be assigned to cities and the knapsack capacity and the renting rate need to be set.

In this section I will explain how these instances are generated and I included some critical remark about the way they are generated. I believe that there is still room for improvement regarding these benchmark instances for it to represent the full potential of the hardness of the traveling thief problem.

### 4.3.1 Knapsack Type

As mentioned before the correlation between weight and value of the items can decide the complexity of a knapsack problem [31]. Therefore Polyakovskiy et al. chose to have three different KP types: uncorrelated, uncorrelated with similar weights and bounded strongly correlated. The fact that the profit weight ratios are close to each other with strongly correlated items makes it hard to solve (Section 3.4).

However what is hard to solve for the knapsack problem is not necessarily hard for TTP. In TTP this profit weight ratio is of less importance since the contribution towards the objective function does not solely dependent on the profit but also on the distances of cities. We can sort items according to a ratio dependent on profit and weight and distance. This is done by almost all current TTP heuristics in the form of a greedy packing heuristic (Section 4.4.1).

It would be interesting to also have instances where items are generated to have a strong correlation between profit and weight and distances. I predict these instances are much harder to solve since the ordering contains less information as is the case for strongly correlated instances of normal knapsack problems.

### 4.3.2 Item Distribution and Capacity Constrain

Every city of a problem instance gets a constant amount of items depending on the item factor. Every instance has an item factor $F_i \in \{1, 3, 5, 10\}$.

This is an interesting choice made by the authors because one could imagine that non-uniform distribution of items over the cities would lead to interesting examples. Simply varying the amount of items that a city gets might not lead to something interesting besides more decision variables. In fact it might even lead to less interesting problems since a higher percentage of cities have items with a good profit weight ratio. In other words, if there aren't any cities which deserve priority then the order in which cities are visited might not matter that much for the best packing plan. In turn this makes the subproblems less

dependent on each other. This is just speculation for now but it seems evident the interdependence chances for incredible high item factors.

The capacity is set as a fraction of the total weight, where the fraction is chosen from the set $\{\frac{1}{11}, \frac{2}{11}, ..., \frac{10}{11}\}$. Resulting in 10 different capacity categories.

### 4.3.3 Renting Rate $R$

As mentioned before the value of the renting rate $R$ is crucial for the interdependence of the problem. It must be chosen in such a way that both the profit of an item and the travel time contribute an equal amount to the objective function. The contribution towards the objective function should be roughly the same as to ensure that one does not dominate the other. This balance is enforced by setting the renting rate as:

$$R = \frac{\text{optimal value KP}}{\text{aprox. optimal value TSP}}$$

Where the optimal value of the tour is approximated by a solution found via the Chained Lin-Kernighan heuristic.

The renting rate is a vital aspect in how hard the problem is. Wu, Polyakovskiy, and Neumann [70] studied the effect of the renting rate of the unconstrained packing while traveling problem. They looked at ranges of renting rates in which the decision of picking up an item is non-trivial. Non-trivial here means that it is not the case that is always profitable to pick up a certain item (or not). More general, the renting rate decides what amount of influence the profit of items or distance traveled has on the objective function. Polyakovskiy et al. [56] choose to set $R$ to a specif value dependent on CLK tour and the optimal packing plan. However by not varying $R$, the instances or some items of those instances might become trivial. Moreover defining the fitness function as a solution found by CLK could in itself create a bias towards similar solutions found by CLK which is a common practice in literature [13, 14, 16, 36].

### 4.3.4 The Generated Benchmark

The benchmark consists of 9720 different TTP instances. Build on 81 TSP instances, with 3 KP types, 4 item factors and 10 capacity categories.

The actual benchmark published online[1] weirdly enough is not created in the same way as described. If you look at the actual benchmark instances there are a few differences:

- They claim the knapsack items for the category bounded strongly correlated are constructed the following way: The weight $w_{ik}$ is chosen randomly in the interval $[1, 10^3]$. But if you look at the data it becomes evident that weight values as high as 4000 exists (Figure 12).

[1] `https://cs.adelaide.edu.au/~optlog/CEC2014COMP_InstancesNew/`

- They claim the profit for the items of the knapsack type bounded strongly correlated is set to $p_{ik} = w_{ik} + 100$. But in the data the profit is also defined by adding 200, 300 and 400 to the weight (Figure 12). A little bit sloppy but this could also influence the structure of the problem. The items that get +400 instead of +100 are favorable.

- Items in all of the 9720 instances are created with only three different sets of knapsack items for the three knapsack types. If this was intended they could have just provided 3 knapsack instances and 91 TSP files instead of 9720 different files![2]

  Luckily this doesn't imply that the instances have the same structure since the decision to pick up an item depends on the tour and that differs between instances.

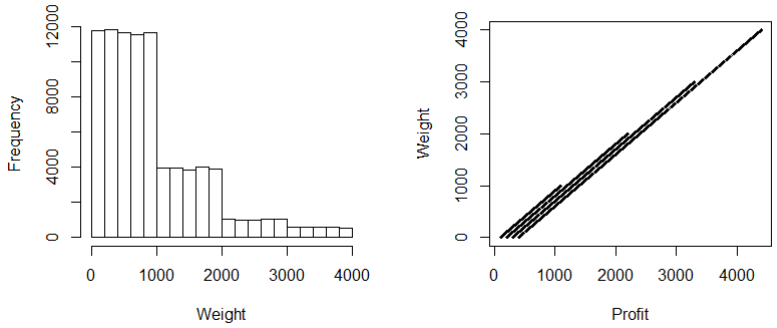


Figure 12: Frequency (left) of items with a certain weight and a plot of the item profit and weight (right) of the TTP-instance $pla85900\_n85899\_bounded\text{-}strongly - corr\_01$

#### 4.3.5 Conclusion Benchmark

The differences between the proposed and the actual benchmark do not matter that much, the instances are still hard and most certainly interesting but I think it is important to know these differences exist.

Most literature about the TTP use these benchmark instances to compare their algorithm with others and this will probably not change any time soon. As I mentioned in the previous sections, it would be interesting to look at instances that are not included in the benchmark. For example, those with a ratio that have a strong correlation between profit and weight and distance,

[2]The only thing that would be missing is the renting rate. That needs the be specified for the 9720 differnt instances.

those with varying renting rate and those where items are spread over cities in a non-uniform way.

## 4.4 Single Solution Algorithms

### 4.4.1 DH and CoSolver

Bonyadi et al. [8] provided two algorithms to solve TTP: Density-based Heuristics (DH) and CoSolver. DH is a greedy packing heuristic. First the tour is generated with Chained Lin-Kernighan heuristic [2]. After the tour is generated, DH fills the knapsack according to a given ordering. This procedure is common for all greedy packing heuristics in the literature [56, 16, 14, 15, 41, 37, 63, 62, 29, 33] and is described in Algorithm 2. The main difference between packing heuristics is either the order in which the items are processed or the mechanism that includes an item into the packing plan. Some heuristics use a fitness evaluation and others an approximate fitness evaluation.

The ordering for DH is based on a score that is calculated by:

$$score_1(I_{ik}) = p_{ik} - R * t_{ik}$$

$$t_{ik} = \frac{d_i}{v_{max} - \frac{v_{max} - v_{min}}{C} * w_{ik}}$$

where $I_{ik}$ is the item $k$ at city $i$, $p_{ik}$ and $w_{ik}$ are the profit and weight of item $I_{ik}$ and $d_i$ is the total remaining length of the tour at city $i$. DH is described in Algorithm 3.

CoSolver is an algorithm that decomposes TTP into two subproblems. It then proceeds to solve these subproblems and exchanges information between the two subproblems. In the end the solutions of the subproblems are combined to form a TTP-solution. How these subproblems are solved isn't made explicit. They state that the TSP-subproblem is solved by an exact algorithm and the KP-subproblem is relaxed and solved by dynamic programming.

Results show that on most test instances CoSolver outperforms DH. Tests were performed on rather small instances $n \leq 25$ possibly because Cosolver has an exact subroutine. According to the authors the fact that DH performs worse than CoSolver confirms that considering the interdependence is beneficial in solving multi-component problems.

### 4.4.2 SH, RLS, (1+1)-EA

Besides providing a TTP benchmark suite, Polyakovskiy et al. [56] also defined some simple algorithms: Simple Heuristic (SH), Random Local Search (RLS) and (1+1) Evolutionary Algorithm ((1+1)-EA).

The main difference between these algorithms is their name. They all use the same Chained Lin-Kernighan heuristic to construct an initial tour and then apply a greedy or local search to the packing plan. RLS and (1+1)-EA are described in Algorithm 4. They both iteratively mutate the packing plan and save the changes if it leads to a better solution. SH is similar to DH. The only

difference is that DH calculates the actual increase of the addition of an item where SH approximates this by calculating the total gain of adding an item on an otherwise empty tour.

**Algorithm 2** General approach for a greedy packing, heuristic FillSack

```
1: function FillSack(L)                                   ▷ L is a sorted list of items
2:     Y ← ∅                                              ▷ initial knapsack empty
3:     W ← 0                                              ▷ initial weight set to zero
4:     k ← 1
5:     while W < C ∧ k ≤ m do
6:         if W + w_{L_i} ≤ C ∧ Z(Π, Y) > Z(Π, Y ∪ {L_i}) then
7:             Y ← Y ∪ {L_i}
8:             W ← W + w_{L_i}
9:         end if
10:        k ← k + 1
11:    end while
12:    return Y
13: end function
```

**Algorithm 3** Simple Heuristic/ Density Heuristic SH

```
1: function SH
2:     Π ← get tour with CLK
3:     for every item I_ik compute score_1(I_ik)
4:     L ← acquire list by sorting items according to scores
5:     Y ← FillSack(L)
6:     return (Π, Y)
7: end function
```

#### 4.4.3 JNB and J2B

El Yafrani and Ahiod [13] proposed two hill climbing algorithms, JNB and J2B, with two different mutation operators. JNB uses a combination of BitFlip and a swap between consecutive cities and J2B uses a combination of BitFlip and 2-opt. The two mutations have a combined neighborhood of $O(n * m)$ and $O(n^2 * m)$. This together with a fitness evaluation of $O(n + m)$ makes a single mutation computationally intensive resulting in a high running time for even small instances. In part they solve this by restricting the neighborhood of 2-opt to edges found in a Delaunay triangulation (this was previously done by Mei, Li, and Yao [36]).

They also experimented with initial tours and concluded that using a good TSP-tour as initial solution resulted in far better results compared to using a random tour as initialization. They found that JNB and J2B performed better than (1+1)-EA and RLS when it could finish within the time limit.

**Algorithm 4** Random Local Search RLS and (1+1)-EA

```
1: function RLS
2:     Π ← get tour with CLK
3:     Y ← ∅                                          ▷ inital knapsack empty
4:     k ← 1
5:     while k < 10000 do
6:         In case of RLS:
7:         Y' ← invert status of random item
8:         In case of (1+1)-EA:
9:         Y' ← invert status of each item independently with probability 1/m
10:        if Z(Π, Y) < Z(Π, Y') ∧ w(Y') ≤ C then
11:            Y ← Y'
12:        end if
13:        k ← k + 1
14:    end while
15:    return (Π, Y)
16: end function
```

#### 4.4.4 S1-S5 and C1-C6

Faulkner et al. [16] provided a series of algorithms all based around their packing heuristic that called PackingIterative. PackingIterative uses a subroutine Pack. Pack is a greedy packing heuristic and adds items based on a score that is defined in the following way:

$$score_2(I_{ik}, \alpha) = \frac{p_{ik}^{\alpha}}{w_{ik}^{\alpha} \times d_i}$$

where $p_{ik}$ and $w_{ik}$ are the profit and weight of item $I_{ik}$ and $d_i$ is the total remaining length of the tour at city $i$. The exponent $\alpha$ changes the impact the variables have on the score. The exponent therefore influences the order in which items are picked up. The success of the heuristic depends on a good value for $\alpha$. Pack is described in Algorithm 5.

**Algorithm 5** Packing Routing Pack

```
1: function PACK(Π, α)
2:     for every I_ik compute score_2(I_ik, α)
3:     L ← acquire list by sorting items according to scores
4:     Y ←FillSack(L)
5:     return Y
6: end function
```

Calculating the fitness score $Z(\Pi, Y)$ is computationally expensive. Therefore Faulkner et al. only compute the fitness after multiple items are added and backtrack if the score became worse. PackingIterative produces different picking plans for variable values of $\alpha$ in order to find the best value for $\alpha$. The

exponent $\alpha$ starts at a certain value and after each iteration the algorithm checks whether $\alpha$ should be increased or decreased. After some iterations it returns the best packing plan found. PackingIterative is described in Algorithm 6[3].

In their paper, Faulkner et al. propose 11 different variants of the same algorithm called S1-S5 and C1-C6. All algorithms begin with a tour $\Pi$ found by Chained Lin-Kernighan heuristic. After which the tour is kept fixed and PackingIterative is performed on the packing plan. Depending on the variant of the algorithm, multiple local searches are performed or the algorithm is started again. The local searches included in the paper are: BitFlip, Insertion and (1+1)-EA.

They also propose an algorithm that is based on the mixed integer programming procedure from Polyakovskiy and Neumann [54]. Again CLK is used to get a tour after which the optimal packing plan is approximated with mixed integer programming.

The variant S5 performs best on a large variety of instances and is used as comparison for many algorithms [14, 15, 33, 64]. It is a multistart algorithm that repeats S1. S1 performs CLK and PackingIterative without any further local search.

**Algorithm 6** Iterative Packing Routine PackIterative

```
1: function PackIterative(Π, α, δ, maxIterations)
2:     P_l ← Pack(Π, α − δ)
3:     P_m ← Pack(Π, α)
4:     P_r ← Pack(Π, α + δ)
5:     i ← 1
6:     while i ≤ maxIterations do
7:         if Z(Π, P_l) > Z(Π, P_m) ∧ Z(Π, P_l) ≥ Z(Π, P_r) then
8:             P_m ← P_l
9:             α ← α − δ
10:        else if Z(Π, P_r) > Z(Π, P_m) ∧ Z(Π, P_r) > Z(Π, P_l) then
11:            P_m ← P_r
12:            α ← α + δ
13:        end if
14:        δ ← δ/2
15:        P_l ← Pack(Π, α − δ)
16:        P_r ← Pack(Π, α + δ)
17:        i ← i + 1
18:    end while
19:    return P_m
20: end function
```

[3]This pseudocode differs slightly from the pseudocode provided by Faulkner et al. [16] in order to remove redundancy

#### 4.4.5 CS2SA

In another paper by El Yafrani and Ahiod [14] two algorithms are introduced. A single solution algorithm inspired by CoSolver called CS2SA and a population based algorithm MA2B. I will discuss MA2B in the next section to put it in perspective with other population based algorithms. CS2SA works the following way: First it initializes the tour with CLK and it initializes the packing plan with greedy packing heuristic similar to the heuristic of Mei, Li, and Yao [36]. Then it keeps trying to improve the tour by 2-opt with a reduced neighborhood of the Delaunay triangulation. After which it tries to improve the packing plan with simulated annealing. This process of improving tour and then the packing plan is repeated until no improvements are made. They found that CS2SA is competitive with S5 and MATLS and that it performed better on instances with high knapsack capacities.

Table 1: Overview of single solution algorithms.

| Algorithm | | $\Pi$ Local Search | $Y$ Local Search | Packing Heuristic |
|---|---|---|---|---|
| DH | [8] | - | - | DH |
| SH | [56] | - | - | SH |
| RLS | [56] | - | Flips random bit | - |
| (1+1)-EA | [56] | - | Flips bit with prob $\frac{1}{m}$ | - |
| JNB | [13] | Consecutive swap | BitFlip | - |
| J2B | [13] | 2-opt | BitFlip | - |
| S1 | [16] | - | - | PackingIterative |
| S2-5,C1-C6 | [16] | Insertion | BitFlip, (1+1)-EA | PackingIterative |
| MIP | [16] | - | - | Approximated MIP [54] |
| CS2SA | [14] | 2-opt | Simulated Annealing | Heuristic from [36] |

### 4.5 Evolutionary Algorithms

#### 4.5.1 MA, CC and MATLS

Mei, Li, and Yao [37] proposed two population based algorithms. A cooperative co-evolution algorithm (CC) similar to CoSolver but with population and a memetic algorithm (MA).

CC keeps two populations containing partial solutions to the subproblems, a population with packing plans and a population with tours. It iteratively progresses the packing plan with BitFlip, Exchange and the one-point crossover OPX. It iteratively progresses the tour with 2-opt and order crossover OX. In order to do a fitness evaluation for the partial solutions a collaborator from the other population is needed. Only the $k$ best members of a population are considered to be a collaborator. A fitness evaluation of a partial solution is done with all the $k$ best members of the other population and the best evaluation of those $k$ members is used.

MA is a memetic algorithm that initializes the population randomly. After which for each generation the following happens:

1. Two members of the population are randomly chosen. $P_1 = (\Pi_{p_1}, Y_{p_1})$ and $P_2 = (\Pi_{p_2}, Y_{p_2})$
2. An offspring $C$ is created by doing order crossover on the tour and two-point crossover on the packing plan.

$$C = (OX(\Pi_{p_1}, \Pi_{p_2}), OPX(Y_{p_1}, Y_{p_2}))$$

3. With a certain probability a local search is performed on $C$. The local search consists of a best improvement hill climbing procedure which consecutively makes one improvement with one pass of 2-OPT then BITFLIP and at last EXCHANGE.
4. After a sufficient number of offspring has been created a truncation selection is performed.

They found that MA outperforms CC and claim this is because CC optimizes the subproblems separately and MA solves the problem as a whole. Further more they claim this illustrates the importance of considering the interdependence of the subproblems. In my opinion this difference could also be explained by the fact that intermediate partial solutions of CC are local optima with regard to 1 or 2 neighborhoods and the intermediate solutions of MA consider 3 neighborhoods. A local optima with 3 instead of 1 or 2 neighborhoods will probably be of higher quality.

In another paper by Mei, Li, and Yao [36] they propose a computational efficient version of their memetic algorithm with a two-stage local search (MATLS). That is partly reminiscent of S5. First a population is created and initialized. The tour is initialized with CLK or a minimal spanning tree heuristic. The packing plan is initialized by a greedy packing heuristic similar to SH. The difference is the fitness evaluation. In MATLS it is approximated by the worst possible and by the expected gain. The worst possible gain is the net increase of an item where the assumption is made that items are picked up in the worse case possible. Worst case is when all items included in the knapsack are picked before the city of the current item. The expected gain is the assumption that all previously picked up items are picked up along the tour with an even distribution.

In each generation OX is performed on the tours after which 2-OPT is done with a reduced neighborhood by the Delaunay triangulation. The fitness evaluation of TSP is used which takes constant time instead of $O(n + m)$. For each new member in the population the packing plan is created by the same greedy packing heuristic I mentioned before.

In their paper they compare their algorithm to (1+1)-EA and RLS and found they have better results.

Following up on this paper Mei et al. also experimented with evolving the packing heuristic with genetic programming [38]. Interesting idea that had competitive results.

### 4.5.2 MA2B

In the same paper where El Yafrani and Ahiod [14] introduced CS2SA they also proposed a memetic algorithm MA2B. The population is initialized in the

following way: First the tour is created by CLK, then a packing plan is generated based on a heuristic of Mei, Li, and Yao [36] after which a restricted local search is applied with maximum of 50 passes. They use the same local search as with CS2SA.

MA2B uses the double bridge move as mutation operator and MPX as crossover. The reasoning behind the use of MPX is that it is a disruptive crossover. They argue that this is preferred in memetic algorithms. Preliminary tests (not explicit in the paper) confirmed that MPX performed better than ERX.

No crossover was directly used on the packing plan, the status of the offspring was inferred from MPX and the packing plan. If the city containing the item is inherited from the first parent then the state of the item is also inherited from the first parent.

$$y_{ik_{Offspring}} = \begin{cases} y_{ik_{Parent1}}, & \text{if city } i \text{ was inherited by Parent1} \\ y_{ik_{Parent2}}, & \text{otherwise} \end{cases}$$

They also compared their two algorithms with S5 and MATLS and concluded that both algorithms showed competitive performance. No algorithm dominated the others on all instances. The memetic algorithms performed better on smaller instances and S5 and CS2SA better on larger instances. They also found that for large instances MATLS spends the majority of the time on its initialization. Furthermore they state that it is a little unusual that greedy algorithms perform very well for many instances and maybe landscape study could give insight in why this is the case. I agree with this sentiment and will investigate this in my thesis.

### 4.5.3 Other Memetic Algorithms

In [29, 41, 62] rather similar memetic algorithms were proposed with as notable difference the crossover operator: order-based crossover [62] and partially mapped crossover [29, 41]. Lourenço, Pereira, and Costa [29] counted the number of edges that are different between the optimal TSP tour and the tour of the best found TTP solution. They found that in all cases but one, the difference exceeded 50% of the edges.

### 4.5.4 Ant Colony Optimization

Wagner [63] proposed an algorithm that finds a TTP solution with the help of swarm intelligence, a MAX-MIN ant colony optimization algorithm called MMAS. Ants construct a tour choosing the next city based on a probability that is proportional to the pheromone associated with edges between cities. After a tour has been found, a TSP specific local search is performed, 2-OPT, 2.5-OPT or 3-OPT. Just like in MATLS this is a local search where the fitness evaluation is done without considering the KP-part of TTP. Only distances between cities are considered. After a tour has been established a packing plan is constructed

Table 2: Overview of choices for population based algorithm. "LS TSP-evaluation" means that they used a local search with a TSP specific fitness evaluation

| Algorithm | | Crossover Tour | Crossover Packing Plan | BitFlip | Exchange | 2-opt | Packing Heuristic | Mutation | Intialization Tour | LS TSP-evaluation |
|---|---|---|---|---|---|---|---|---|---|---|
| CC | [37] | OX | OPX | Yes | Yes | Yes | - | - | - | - |
| MA | [37] | OX | OPX | Yes | Yes | Yes | - | - | - | - |
| MATLS | [36] | OX | - | Yes | - | - | Yes | - | CLK | Yes |
| MA2B | [14] | MPX | - | - | - | Yes | Yes | Yes | CLK | - |
| MCGA | [62] | OBX | 3PX | Yes | - | Yes | Yes | Yes | CLK | - |
| EA | [29] | PMX | - | - | - | - | Yes | Yes | - | - |
| Hybrid | [41] | PMX | - | Yes | - | Yes | Yes | Yes | NNH | - |
| MMAS | [63] | - | - | Yes | - | - | Yes | - | - | Yes |

using PackIterative from Faulkner et al. [16]. As an optional step local search is performed that does consider the total fitness function: (1+1)-EA, one pass of Insertion and one pass of BitFlip. After which the pheromone trails are updated based on how the ants perform. Stepwise[4] this would look like this:

1. Construct tour using ants.
2. Perform TSP-specific local search on tours: 2-opt, 2.5-opt or 3-opt.
3. For every tour create packing plan with PackIterative.
4. Perform TTP-specific local search on tour: (1+1)-EA, one pass of Insertion and one pass of BitFlip.
5. Update pheromone trail.

Wagner shows that his ant colony optimization algorithm picks longer tours than the approximate algorithms S1-5 and C1-6. Longer tours but with better results. MMAS outperforms most current approaches on instances with up to 250 cities and 2000 items [63, 64]. Wagner claims this is because it focuses less than existing approaches on good TSP tours, but more on good TTP tours. But this is not apparent from his approach at all. Actually, together with MATLS they are the only population based approaches that have TSP-specific local search (Table 2). I suspect the good results are due to the fact their tour is not generated by CLK as is the case with most single solution algorithms. MMAS generates many different "weaker" local optima that are not found by CLK. Those other optima that CLK can't find might provide better tours and thus MMAS has better results.

[4]The order of steps is different in Wagner's paper [63]. In his paper step 2 and 3 are reversed but this is not the control flow of the source code available online `https://cs.adelaide.edu.au/~optlog/research/ttp/2016ants.zip`.

### 4.5.5 Hyper Heuristics

One interesting approach to solve TTP is with a hyper-heuristic. El Yafrani et al. [15] have done this and a similar paper by Martins et al. [33]. A population of encodings is kept where each individual of the population represents a combination of multiple low level heuristics (LLH). The LLH's are all applied in succession after an initial tour via CLK and a packing plan via PackIterative is created. The LLH's are either local searches or disruptive operators. The different local searches are: BitFlip, simulated annealing [14] on packing plan and 2-opt. The disruptive operators are: random 2-opt swap, the double bridge move and bit flips on a percentage of items.

El Yafrani et al. [15] use genetic programming and Martins et al. [33] sample new solutions using an estimation of distribution algorithm. Martins et al. also approximates the fitness evaluation with a radial basis function network. Their algorithms are competitive with S5, MMAS and MA2B.

## 4.6 Exact Approach

Wu et al. [71] proposed some exact approaches, one with dynamic programming, one with branch and bound search and one with constraint programming. Given the hardness of the problem these algorithms' time complexity grows exponentially with the number of cities. On small samples ($n \leq 20$) the optimal solution can give insight in how algorithms perform. They found that PackingIterative of S5 could be still be a little bit improved by using an exact packing approach and they found that MA2B has an outstanding performance across all instances with a high reliability.

## 4.7 Algorithm Selection and Comparison

Wagner et al. [64] studied the applicability of algorithm selection to the traveling thief problem and among other things compared the performance of 21 algorithms on all 9720 instances for 10 minutes! The algorithms they compared were: SH, DH, (1+1)-EA, RLS, S1-S5, C1-S6, MALTS, CS2SA and four different configurations of MMAS. The results are available online[5] and can serve as a great tool for comparison.

On average S5 had the best ratio to best found solution. It is interesting to note however the compared algorithms are really similar. All algorithms except (1+1)-EA used a greedy packing heuristic and all algorithms except MMAS used CLK as (initial) tour. They also found that CS2SA found best solutions on a number of instances while having the worst performance on average.

Unfortunately from the source code[6] it becomes evident that El Yafrani and Ahiod [14] have made a rounding error in the fitness function. The distance between cities were rounded down instead of up. This makes a huge difference

[5] `https://cs.adelaide.edu.au/~optlog/research/ttp/160624-21algs-scmatls-1run.csv`

[6] `https://github.com/yafrani/ttplab`

on some TTP instances where the x- and y-coordinates have small integer values. Therefore the results and conclusions in their paper and others [64] are unreliable. El Yafrani and Ahiod are not the only ones to make this mistake, I found a similar error with the work of Vieira et al.[7] [62] and Moeini, Schermer, and Wendt [8] [41].

## 4.8 Conclusions

### 4.8.1 Trends

In the four years since the problem has been formulated a number of algorithms have been proposed. Although some strategies probably had different goals in mind, there are a few trends to be seen:

- Use of TSP-specific local search with neighborhood reduction strategies.
- Use of a greedy packing heuristic to generated a (initial) packing plan.
- Use of local search 2-OPT, INSERTION, BITFLIP and EXCHANGE.
- Use of memetic algorithms with OX, PMX or MPX as crossover for the tour.

For the trends there is no real justification for using these operators/strategies besides some comparison between different algorithms or previous findings from the TSP literature.

### 4.8.2 Interdependence

Some papers claim that one of their algorithms performs better since it considers the interdependence or TTP-specific tours more than other algorithms. COSOLVER is better than DH, MA is better than CA and MMAS performs better than S1-S5, C1-C6 and MATLS for small instances. The authors of these papers claim this is the case because their algorithm considers important TTP-aspects while others don't. These claims however are only backed up by their good performance relative to others and not by any other analytic argument or other empirical study.

What does it mean for an algorithm to consider the interdependence of TTP? The interdependence, to put it simple, is the fact that if you choose certain items then in turn cities containing these items will likely be at the end of the tour. If you have cities at the end of the tour then the items belonging to these cities are more likely to be in the packing plan since the travel time to the end of the tour is shorter.

It seems that the greedy packing heuristic is a strategy that takes into consideration the tour aspects when forming the packing plan. Using the fitness evaluation could be seen as considering the interdependence of TTP. But these two things are present in almost all proposed algorithms. COSOLVER, MA and

[7] `https://github.com/DanielKneipp/GeneticAlgorithmTravelingThiefProblem/blob/master/ttp/city.cpp`

[8] In email-correspondence with Moeini, he confirmed there was a rounding error in the source code.

MMAS obviously do some things better, hence the better fitness, but the cause of these differences are attributed to considering interdependence or solving TTP as a whole without much justification.

### 4.8.3 Focus on Large Instances

Some papers [8, 37, 71] have focused on small size instances ($n \leq 100$), some papers [13, 15, 16, 33] focus on mid size instances ($n \leq 1000$) and some papers [14, 16, 36, 56, 64] focus on large size instances ($n \leq 85900$). I think its better to focus on small or mid size instances because of two reasons:

- The motivation behind the introduction of TTP was to close the gap between real-world problems and research [7]. Bonyadi, Michalewicz, and Barone argue that one difference is how complexity is defined. Complexity in benchmark problems refer to the scale of the problem. The complexity of real-world problems on the other hand get their complexity from the interdependence between their components rather than number of objectives or the size of the problem. It therefore seems odd that a lot of research has focused on large instances given the motivation behind the TTP.

- Trying to solve large instances that are already hard for TSP with a fitness evaluation function that takes $O(n)$ times as long seems really hopeful. That is even without considering the extra complexity from the knapsack problem and the interdependence between the two. If this could be achieved, this could probably be carried over to TSP but my guess is that improving state-of-the-art TSP algorithms is really hard. Therefore I think it would be more interesting to understand how hard the problem is and what this interdependence actually adds to the complexity. This instead of cutting corners with Delaunay triangulation, approximate fitness evaluation [14, 16, 38] or solving the two subproblems independently [16, 36, 56] in order to get any answer for really large instances.

# 5 Fitness Landscape Analysis

The effectiveness of using heuristic optimization for NP-Hard combinatorial optimization problems can be shown empirically. Either by producing good solutions or by comparing them to other approaches. A much harder problem is to show why and how heuristic optimization works [39, 40].

Fitness landscape analysis tries to answer these questions or at least gives us some insight in how hard the problem is, how solutions are related, how operators transform solutions and which parameters are best suited for the problem at hand.

According to Jones et al. [24], a landscape is a way to look at some aspects of a complex process. Landscape therefore is a metaphor, a tool, that can help us understand aspects of the complex process of finding the optimal solution in a discrete search space. Visualizing the search space as a 3D landscape where the objective is finding the highest peak is appealing. Jones et al. argue that simple properties of a fitness landscape rely heavily on physical three dimensional landscapes like peaks, ridges and valleys. Although the metaphor can be useful it could also be harmful as it is not clear if these properties scale up to search spaces with higher dimensions.

Nonetheless fitness landscape analysis has been proven to be a valuable tool and in this chapter I want to discuss some fitness analysis I tend to use to understand various operators and strategies to solve TTP.

## 5.1 Formal Definition Fitness Landscape

A fitness landscape is a tuple $(X, f, d)$. Where $X$ is a set containing all possible solutions, the search space. $f : X \to \mathbb{R}$ is a function from solutions $\in X$ to a fitness value $\in \mathbb{R}$. And the third element is a notion of how they are related. For this we use a distance measure, $d : X \times X \to \mathbb{R}$. $d$ can be defined with the neighborhood of a particular operator (2-OPT for example) or as the distances between two representations of solutions (hamming distance for example). It is important to note that this implies that every operator, every representation and every problem instance has its own fitness landscape [24, 53].

A fitness landscape can also be seen as a graph $G = (X, E)$ where the nodes are equal to the search space, $X$, and the edges are defined as $\{\langle x, y\rangle \in X \times X \mid d(x, y) = d_{min}\}$ where $d_{min}$ is the minimal distance between two solutions [39].

## 5.2 Ruggedness, Autocorrelation and Correlation Length

One important property seems to be the ruggedness of a landscape. The more rugged a landscape is, the harder it is to navigate. Autocorrelation is a measure first investigated by Weinberger [65] and is used to quantify the ruggedness of a landscape. Although this measure can be useful, it only gives a basic view of the landscape and cannot always predict the hardness of a problem [24].

Autocorrelation measures the correlation of fitness between neighboring solutions. One way to measure it is by repeatedly picking a random point $x \in X$

and measure the correlation between all neighboring solutions. Another way to measure it is by doing a random walk through the landscape and calculate the correlation between succeeding solutions [40].

To calculate the autocorrelation with a random walk:

- Start at a random solution $x_1 \in X$, a random point on the landscape
- Make a random walk $[x_1, ..., x_n]$ with regard to some neighborhood
- Calculate R(s) with regard to some step size $s$

$$R(s) \approx \frac{1}{\sigma_f^2(n-s)} \sum_{i=1}^{n-s} (f(x_i) - \overline{f})(f(x_{i+s}) - \overline{f})$$

Related to autocorrelation is the correlation length $l$, the largest step size for which there is correlation and is defined as $l = -\frac{1}{ln(|R(1)|)}$. High correlation and a long correlation length indicate a smooth landscape where a low correlation and a short correlation length indicate a rugged landscape [39].

Merz [39] proposed another interesting method. Instead of doing a random walk, one could also walk towards another optima. The intermediate values of the walk between local optima could be better than on a random walk. This could indicate that recombination can be very effective. This only holds for recombinations and mutations that are similar. For example: for BitFlip and 1-point crossover this makes some sense but not for 2-opt and MPX. All children of the 1-point crossover can be reached from a short random path from one parent to the other. 2-opt and MPX are too dissimilar, one reverses subtours and the other injects subtours.

## 5.3 Crossover Correlation

The crossover correlation coefficient is a measure that tries to capture the relation between parent and child of a certain crossover in term of their fitness. It was first introduced by Manderick [30] and is defined the following way:

$$\rho_{op}(f_p, f_c) = \frac{Cov(f_p, f_c)}{\sigma(f_p)\sigma(f_c)}$$

where $f_p$ and $f_c$ are fitness of the parent and child. A higher correlation coefficient suggests better preservation of information during inheritance [34].

Mathias and Whitley [34] used this measure for different TSP crossovers in three different scenarios. One without local search, one where 2-opt is applied to the initial population and one where 2-opt is applied to the initial population and produced offspring. They found that ERX and MPX have similar correlations in the first case and MPX has higher correlations when 2-opt was applied in the other cases.

## 5.4 Distance to Global Optimum

The fitness distance correlation coefficient (FDC) was proposed by Jones and Forrest [25] and can be used to measure the difficulty of a problem for certain metaheuristics. A high fitness distance correlation coefficient indicates that an optima with a higher fitness is more likely near the global optima than optima with a lower fitness. These properties are useful for local search. For example, if a landscape has a high correlation then iterated local search can navigate it quite easily since it iteratively moves towards better optima and hopefully towards the global optimum.

The fitness distance correlation can be estimated by:

$$FDC(f,d) \approx \frac{1}{\sigma(f)\sigma(d)} \frac{1}{n} \sum_{i=1}^{n} (f_i - \overline{f})(d_i^{opt} - \overline{d^{opt}})$$

Where $d_{opt}$ is the distance to the global optimum or a global optimum in case there are many. According to Jones and Forrest the problems they studied could be grouped into roughly three classes:

- $FDC(f,d) \geq 0.15$ misleading landscape
- $-0.15 \leq FDC(f,d) \leq 0.15$ difficult landscape
- $FDC(f,d) \leq -0.15$ straightforward landscape

For TSP there exists a correlation between the fitness of local optima and the distance to the global optimum [61]. But similar to autocorrelation this cannot always predict a problem hardness [35] but it can give some insight and the structure of the relationship between fitness and distance can be revealed by a scatter plot [25].

## 5.5 Basin of Attraction

All optima have a basin of attraction. Knowing what the basin of attraction is can be very useful for iterated local search and memetic algorithms. The perturbation, mutation and crossover should be designed such that it can escape the basin of attraction [39] but not jump so far out of the basin that it becomes unrelated to the previous optimum.

The larger the basin of attraction, the higher the probability that a local search can find it. This suggests that a preferable property of a fitness landscape would be that local optima with a low fitness should have a smaller basin of attraction than local optima with high fitness [61]. There are several ways to measure this. Theoretically you could locate all points in the basin of attraction of a local optimum $x_l$ and calculate the sum of probability that a point in the basin of attraction actually goes to $x_l$. For most landscapes locating all points in the basin of attraction of a particular optimum is impractical. One way to approximate it would be to generate a large number of local optima and calculate the probability that one is found.

There is also a nice alternative method design for landscapes which have a high ruggedness and therefore a small basin of attraction. Reversed hill climbing was introduced by Jones, Rawlins, et al. [26] and can calculate the exact probability that hill climbing will attain some point in a landscape. They do this with a hill climber in the reverse direction, going towards worse solutions. All the solutions that can be reached by a reversed hill climber are in the basin of attraction of the starting position. The exact probability that a local optimum will be visited can be calculated. The running time of this algorithm corresponds to the size of the basin of attraction and is therefore unfeasible for a lot of problems.

Tayarani-N and Prügel-Bennett [61] found that the number of local optima of all TSP variants grow exponentially and fitter optima had a larger basin of attraction. The probability of finding the global optimum with 3-OPT decreases exponentially.

## 5.6 Big Valley, Single-Funnel and Multi-Funnel

Supposedly the fitness landscape with respect to 2-OPT for the traveling salesman and many other problems looks like a big valley in which good local optima are clustered around the global optimum and optima with a good fitness are correlated with the distances to the global optimum [6]. The gradient viewed on a coarse level looks like a globally convex structure that leads to the global optimum hence the name big valley [20]. In the literature a related term, single-funnel structure is used [68]. In a single-funnel landscape there is a global structure that dominates the entire search space and causes the best local optima to be concentrated in one region of the search space. A multi-funnel structure on the other hand means that the fitness landscape at the scope of local optima are organized into clusters. A particular local optimum therefore largely belongs to a particular funnel [48]. Finding the best local optima in a funnel is readily done with an iterated local search but escaping it is a much harder task.

Hains, Whitley, and Howe [20] found that the overall search space of TSP indeed appears to display the big valley structure but tours with a fitness close to the global optimum do not have this structure. Tours close to the global optimum have a multi-funnel structure. Ochoa and Veerapen [47] expanded this idea confirming that there indeed exists a multi-funnel landscape and not only near the global optimum but much earlier on. Specifically in their study they found that some instances of the TSPLIB are actually composed of subvalleys or funnels. The funnels are of importance since they are structures of which an iterated local search, in this case CLK, cannot escape. In order to escape such a funnel a restart is needed or a crossover operator which is strong enough to escape the funnel like GPX [20].

To analyze the fitness landscape Ochoa and Veerapen made a local optima network. Nodes are local optima and edges are the starting and end optima after a double-bridge move, the perturbation. To get these nodes and edges they run CLK 100 times with $n$ perturbations. This results in $100n$ (some duplicate) local optima.

The constructed graph besides being aesthetically pleasing shows that there exits multiple funnels of large size. Sometimes these funnels do not contain the global optimum. In [46] they found that for some problems with a multi-funnel structure increasing the perturbation size improved the algorithm. This made the fitness landscape smoother with less funnels and the global optimum more accessible with CLK.

# 6 Research

As mentioned in section 4.8.1 various algorithms have been proposed and there are some trends in these algorithms. In the literature there is no real justification for using these operators/strategies besides some comparison between different algorithms or previous findings from TSP literature. Therefore I would like to research two things in my thesis: investigate and understand these operators/strategies and try to improve them. I would like to investigate the previous point in part by fitness landscape analysis. My research question is:

**How can the use and effect of various operators and strategies in literature of the traveling thief problem be justified, explained and improved?**

## 6.1 Subquestions

To answer this question I have some related subquestions:

1. **Can optimal or near optimal solutions for the packing plan be obtained with a greedy packing strategy?**

   In almost all TTP literature a greedy packing heuristic is used based on distance, weight and profit of items. I would like to research if these greedy packing heuristics can actually find good solutions and if they can why.

2. **What is the effect of using different restricted neighborhoods for local search?**

   In some of the literature the neighborhood of 2-opt is reduced by only using edges that are in the Delaunay triangulation of the graph. This makes the neighborhood of 2-opt $O(n)$ instead of $O(n^2)$ but how does it effect the solution quality and how do other neighborhoods compare?

   Neighborhood Reductions:

   - Delaunay triangulation
   - nearest neighbor
   - k-quad-nearest graph

3. **What are the characteristics of the fitness landscape of various operators of traveling thief problem instances?**

   I would like to investigate several measures on fitness landscapes of operators of different instances.

   Measures:

   - Autocorrelation
   - Probability to find the basin of attraction
   - Fitness distance correlation coefficient

   Operators:

   - BitFlip

- 2-opt
- Exchange
- Insertion

Instances:

- Benchmark [56]
- Instances where one of the subproblems is kept fixed

4. **Does there exist a single (big valley) or multiple funnel structure in instances of the TTP?**

   I would like to investigate how local optima are related to each other and find out whether TTP contains a big valley or if it contains multiple funnels. In order to do this I will collect local optima in the context of an iterated local search and research how they are related to each other.

5. **How can state-of-the-art crossover operators for TSP be used for TTP?**

   Crossovers that produce good results for TSP are EAX, PX and GPX2. They cannot directly be used for the TTP since subprocedures of these crossovers are designed for the fitness evaluation of TSP that only uses the distance between cities. I would like to find out if these same subprocedures can be tinkered such that they can be used for the TTP.

6. **How well do different crossover operators perform on TTP?**

   In the literature the following crossovers are used: OX, PMX and MPX. I would like to find out how other crossover methods compare in terms of their performance of instances. Also I would like to gain insight in the crossover correlation with and without performing local search.

   Crossovers: CX, OX, PMX, MPX, EAX, PX and GPX2.

7. **How does using multistart local search, iterated local search and genetic local search compare with state-of-the-art algorithms of TTP?**

   In the end, I would like to propose a multistart local search, iterated local search and genetic local search with the knowledge I have acquired in this research and see how it compares to state-of-the-art of TTP.

## 6.2 Outline Part II

Section 7 is about the complexity improvements of BitFlip, Exchange 2-opt, and Insertion. Section 8 is about the first subquestion. Section 9 answers the second subquestion. Section 10 will answer the third subquestion and will also include an investigation into multistart local search. Section 11 will answer the fourth subquestion and will include an investigation into iterated local search. Section 12 will answer the fifth and sixth subquestions and will include an investigation into genetic algorithms. Section 13 will answer the last subquestion.

In Part II the research questions are answered in part by empirical study performed on a subset of the benchmark instances of Polyakovskiy et al. [56]. Only the instances with a small amount of cities will be used and the item factor will be kept at 1. In part to reduce the size of studied instances, to reduce[9] the running time of the algorithms and because I think a small item factor could be more interesting than higher item factors (see Section 4.3.2).

[9]This reduction is not convenient but necessary for larger instances $n > 1000$ it is impossible to produce a local optimum with the operators BitFlip, Exchange 2-opt, and Insertion in a timely manner.

# Part II

# 7 Complexity Improvements of the Local Search

The local search procedure used in this thesis will be the following: iteratively taking a first improving step sequentially in 2-Opt, Insertion, Bitflip and Exchange (see section 2.1 for an explanation) until no improvement is made.

The complexity of these procedures is as follows:

| | Size Neighborhood | Complexity | Reduced Neighborhood | Reduced Complexity |
|---|---|---|---|---|
| 2-Opt | $O(n^2)$ | $O(n^2(n+m))$ | $O(n)$ | $O(n^2)$ |
| Insertion | $O(n^2)$ | $O(n^2(n+m))$ | no reduction | $O(n^2)$ |
| Bitflip | $O(m)$ | $O(m(n+m))$ | no reduction | no reduction |
| Exchange | $O(m^2)$ | $O(m^2(n+m)$ | $O(m)$ | $O(m(n+m))$ |

How these reductions are accomplished will be explained in the following sections.

## 7.1 2-Opt

The complexity of 2-Opt can be improved in two ways:

- For every city you can precalculate the total picked up value and total picked up weight at that city [36]. This reduces the complexity of the fitness evaluation from $O(n+m)$ to $O(n)$.
- With neighborhood reduction (see section 9) you can reduce the neighborhood from $O(n^2)$ to $O(n)$.

This results in a total complexity of $O(n^2)$.

An additional speedup can be accomplished by the fact that a single 2-Opt swap only affects part of the tour. In 2-Opt a segment of the tour gets reversed. The part in front of the segment and the part after the segment stay the same. Therefore when doing fitness evaluation, you only have to consider the reversed segment. On average this reduces the time by a half.

## 7.2 Insertion

As with 2-Opt the fitness evaluation of Insertion can also be reduced to $O(n)$ by precalculating the weight and profit of items in the cities. But we can reduce complexity of the fitness evaluation even more by using incremental fitness evaluation. Thereby considering the whole neighborhood in only $O(n)$ time.

The size of the neighborhood of Insertion comes from the fact that every city $O(n)$ can be inserted into every position of the tour $O(n)$. A naive implementation has a time complexity of $O(n^2) \cdot O(n) = O(n^3)$. But we can do much better. The fitness of every possible insertion can be efficiently computed in the following way:

1. Place the selected city at the end of the tour
2. Calculate the fitness of this permutation $O(n)$
3. Swap the city with its predecessor and reevaluate the tour $O(1)$
4. Repeat step 3 until all possible insertion locations are evaluated $O(n)$

Step 3 is $O(1)$ since only a small part of the tour changes. Swapping two succeeding positions only alter the cost of 3 edges (see Figure 7.2).

$$f(\Pi') = f(\Pi) - \text{cost of old segment} + \text{cost of new segment}$$

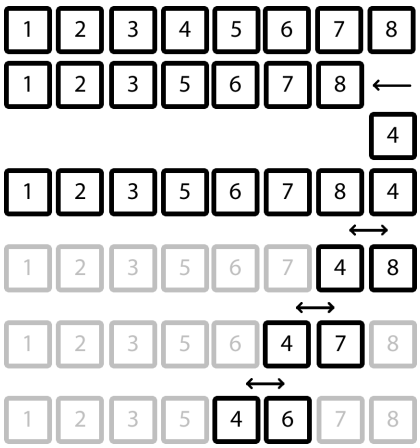


Figure 13: Using incremental fitness evaluation for INSERTION. After placing city 4 at the end of the tour evaluation the tour is $O(n)$. But for all other evaluations $O(1)$ time (the non-gray segment) is needed to calculated the new fitness value.

This makes the total time complexity $O(n^2)$.

## 7.3 Exchange

To improve the complexity of EXCHANGE I have chosen to reduce the neighborhood. Instead of trying $O(m^2)$ possible exchanges I only try promising pairs. The pairs are created the following way:

- For every item $i$ that is not included, create a set $S_i$ containing items $j$ that are included in the packing plan and have a weight equal or greater than $i$. $S_i = \{j | y_j = 1 \wedge w_j \geq w_i\}$
- From $S_i$ find the item $j$ with the minimum loss (or maximum gain) of fitness when the item is not included anymore. $i$ and $j$ form a pair and are considered for EXCHANGE.

Since there are only $m$ items only $O(m)$ exchanges are tried.

# 8 Experimentation: Greedy Packing Heuristic

## 8.1 Introduction

In this section I will investigate greedy packing heuristics that are frequently used in the TTP-literature. As explained in section 4.4.1 they all use a certain ordering in which the items are added to the packing plan. Most of them also use additional optimization to speed up the evaluation of an added item. In this experiment I will disregard optimization and only look at the ordering of the various packing heuristics. The focus of the section is to answer the following research question:

**Can optimal or near optimal solutions for the packing plan be obtained with a greedy packing strategy?**

The three packing orderings under investigation are:

- The Simple Heuristic (SH) from Bonyadi et al. [8] which has the same ordering as DH from Polyakovskiy et al. [56].

$$score_1(I_{ik}) = p_{ik} - R \cdot t_{ik}$$

$$t_{ik} = \frac{d_i}{v_{max} - \frac{v_{max} - v_{min}}{C} \cdot w_{ik}}$$

  Where $I_{ik}$ is item $k$ at city $i$, $p_{ik}$ is the profit, $w_{ik}$ is the weight, $v_{min}$ and $v_{max}$ are the minimum and maximum speed, R is the renting rate and $d_i$ is the distance from city 0 to city $i$ along the chosen tour.

- The Insertion Heuristic (IH) from Mei, Li, and Yao [36].

$$score_2(I_{ik}) = \frac{p_{ik} - R \cdot d_i \cdot \Delta(I_{ik})}{w_{ik}}$$

$$\Delta(I_{ik}) = \frac{1}{v_{max} - \frac{v_{max} - v_{min}}{C} \cdot w_{ik}} - \frac{1}{v_{max}}$$

- Packing Iterative (PI) from Faulkner et al. [16]. Which is the most "advanced" packing heuristic since it searches through multiple packing orderings to find the optimal value for $\alpha$.

$$score_3(I_{ik}, \alpha) = \frac{p_{ik}^{\alpha}}{w_{ik}^{\alpha} \times d_i}$$

## 8.2 Experimental Setup

The tour in the experiments will be kept fixed in order to only investigate the performance of the heuristics and not TTP as a whole. The instances under investigation are:

- $eil51$
- $pr76$
- $bier127$
- $rat195$
- $berlin52$
- $kroA100$
- $ch130$
- $a280$

For every TSP-instance the 3 different knapsack types and 10 capacity constrains $\in \{1, ..., 10\}$ are considered. For every instance I use three different tours: The optimal tour for TSP, the optimal tour in reversed direction and the best TTP tour found after a number of iterations of a simple iterated local search. In total this results in 720 instances.

To evaluate how the packing heuristics perform, I will make a comparison between the heuristics, compare them to a simple local search and compare them to the optimal solution. The simple local search used here only uses the operators that alter the packing plan: first improvement BitFlip and Exchange. The optimal solution of a tour will be calculated with the use of dynamic programming [45]. The three packing heuristics are deterministic and therefore the experiments will not be repeated.

To analyze the heuristics I will rank the heuristics and use an approximation ratio in order to compare different instances with each other. I will also count the number of times the heuristic produces the optimal solution and what percentage of items have the same status as the optimal solution[10].

## 8.3 Results

Of the three packing heuristics Packing Iterative seems to produce the best packing plan in almost all instances (Table 3). Simple Heuristic on the other hand produces the worst packing plans and never outperforms the other two. Insertion Heuristic on occasion performs better than Packing Iterative.

The bad performance of Simple Heuristic can be contributed to the fact that the ordering is only based on the approximated net gain of an item. The net gain is the cost of the added weight (extra time it takes to travel the distance) plus the profit of the item. Insertion Heuristic also approximates the net gain but also divides this by the weight of the item which in turn gives a ratio. This is a crucial aspect for a good ordering.

Remarkably, Packing Iterative finds the optimal solution almost 70% of the time and has an average approximation ratio of 0.999. It seems that greedy packing heuristics can produce really good results. Note however that the success of PI could lie in the fact that it produces multiple (40 in the original paper) packing plans.

Compared to the local search PI on average finds better results but the local search finds the optimal solution more often. The local search seems to have difficulty with the bounded strongly correlated knapsack type (Table 4).

To gain more insight in the ordering of the packing heuristics I visualized them in Figure 14 & 15. Figure 15 shows that the ordering of SH does not really provide the structure to construct the optimal solution. IH provides more

[10]These measures aren't ideal. The maximum score between instances varies a lot and can take on negative values. The percentage of items that have the same status is a distance measure (similar to hamming distance). More about this in Section 10.3.

Table 3: Comparing the three packing heuristics with each other and with the local search.

| | # best packing plan | # opt | approximation ratio | % item opt |
|---|---|---|---|---|
| SH | 5 / 720 | 5 / 720 | 0.678 | 86.23 |
| IH | 171 / 720 | 126 / 720 | 0.975 | 97.26 |
| PI | 717 / 720 | 486 / 720 | 0.999 | 99.25 |
| LS | | 546 / 720 | 0.993 | 98.88 |

structure and with PI the division is the most clear. This is a trend that can be seen for all the instances under investigation (Figure 15 and for even more see Appendix A.1).

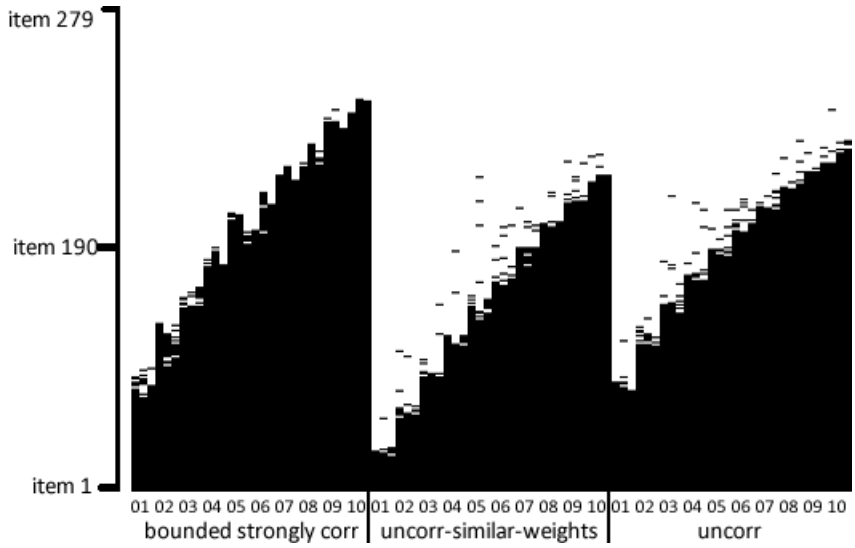


Figure 14: A figure displaying all 90 instances of instance $a280\_n279$. Every column represents a single instance. Every cell represents an item which is black when the item is included in the optimal packing plan. The cells are ordered according to PI (in this case). If the white and black cells are completely separated this implies that the packing heuristic returns the optimal solution.

### 8.3.1 Difference in Knapsack Type

Table 4 shows the performance of the heuristics on different knapsack types.

In terms of approximation ratio SH performs best with the bounded strongly correlated type. This can be explained by the fact that there is a correlation between profit and weight that positively complements the missing profit weight ratio consideration of SH.

Surprisingly for the other two packing heuristics the difference in knapsack type makes no notable difference. The approximation ratio and the frequency

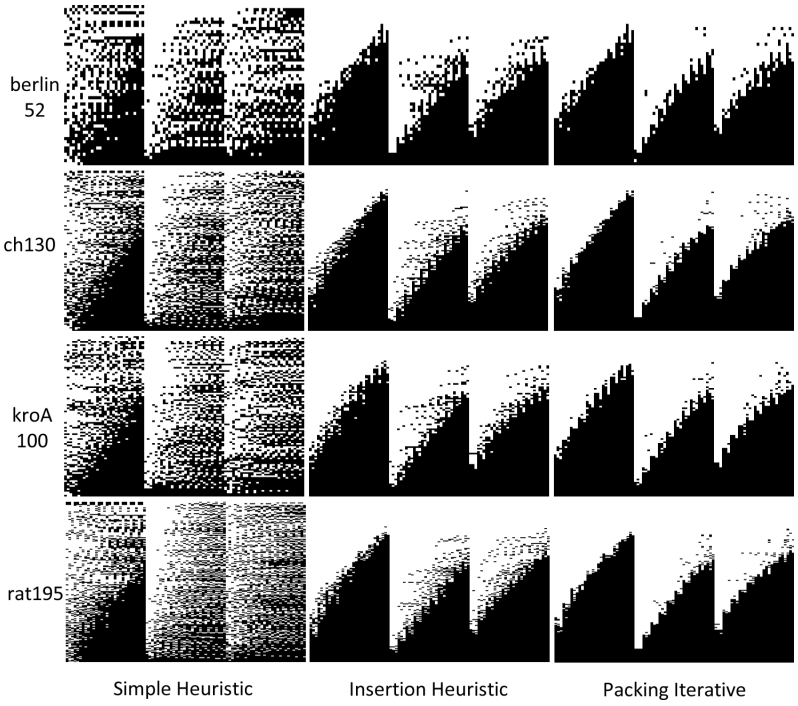


Figure 15: The performance of different packing heuristics visualized. See Figure 14 for an interpretation.

the optimum is found seems to lie really close to each other.

On the other hand the local search seems to have more trouble with the bounded strongly correlated (BSC) type than with the other types. In less than half of the instances the LS finds the optimal solution with BSC. For the other types the optimal solution is found 90% of the time. A possible explanation could be the fact that the local search makes use of Exchange. With the two other types the differences between profit and weight ratios are greater and therefore there are more improving exchanges possible.

## 8.4 Further Experiments

Packing Iterative seems to perform really well. With some further experiments I want to find out why this is and whether a combination of packing heuristic and local search can improve a solution even more.

As mentioned in previous sections the strength of PI can lie in the fact that PI considers multiple packing plans that differ slightly. To test if this is the case I try to achieve the same effect with Insertion Heuristic by adding a small coefficient[11] that is varied slightly in order to produce many packing plans just

[11]The coefficient could be added in a number of places to produce a variety of slightly

Table 4: Difference in knapsack type. #opt is frequency that the optimal solution is found. Fitness is the approximation ratio towards the best found fitness score. % item is the percentage of item that has the same status as the optimal solution of the knapsack

| | bounded strongly corr | | | uncorr-similar-weights | | | uncorr | | |
|---|---|---|---|---|---|---|---|---|---|
| | #opt | fitness | %item | #opt | fitness | %item | #opt | fitness | %item |
| SH | 2 / 240 | 0.8622 | 86.02 | 2 / 240 | 0.5339 | 85.07 | 1 / 240 | 0.6365 | 87.61 |
| IH | 41 / 240 | 0.9752 | 96.15 | 38 / 240 | 0.9652 | 97.49 | 47 / 240 | 0.9852 | 98.13 |
| PI | 152 / 240 | 0.9985 | 98.90 | 151 / 240 | 0.9991 | 99.39 | 165 / 240 | 0.9995 | 99.47 |
| LS | 102 / 240 | 0.9818 | 97.00 | 221 / 240 | 0.9994 | 99.86 | 223 / 240 | 0.9991 | 99.77 |

like PI.

$$score_4(I_{ik}) = \frac{p_{ik} \cdot \alpha - R \cdot d_i \cdot \Delta(I_{ik})}{w_{ik}}$$

Let us call this packing ordering Insertion Heuristic Improved (IHI).

From Table 5 it becomes apparent that IH indeed can be improved by producing multiple packing plans with slightly different orderings and picking the best one. The optimal solution is found three times more often. PI however still outperforms IHI. Therefore the strength of PI lies not only in the fact that it produces many packing plans but the whole procedure seems to perform reasonable well.

Figure 16 shows that varying $\alpha$ with PI changes the packing plan. Moreover the fitness values show an almost concave fitness function making it possible for the uniform binary search approach of Faulkner et al. [16] to work.

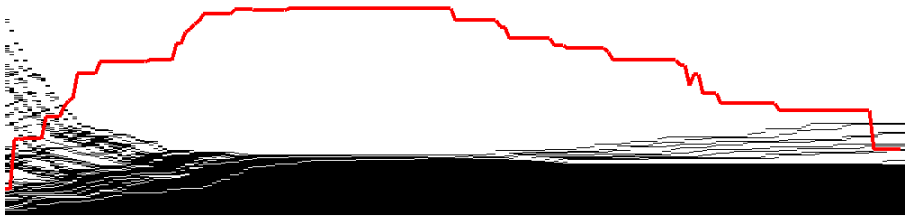

Figure 16: A plot of increasing values for alpha for an instance of *eil_51*. The red line indicates the fitness.

The combination of packing heuristic and local search can find the optimal solution in almost 90% of the instances. The use of the combination is therefore justified especially given the fact that the packing heuristic $O(n \log n)$ is far less computationally expensive than the local search $O(n^2)$.

different packing plans. I have chosen to multiple the profit since it gave the best results.

Table 5: Results from Insertion Heuristic Improved and the combination of packing plans with local search

| | # optimum found | Approximation ratio | % shared with optimum |
|---|---|---|---|
| IHI | 324 / 720 | 0.9951 | 98.70 |
| PI + LS | 640 / 720 | 0.9995 | 99.63 |
| IHI + LS | 610 / 720 | 0.9985 | 99.48 |

## 8.5 Conclusions

**Can optimal or near optimal solutions for the packing plan be obtained with a greedy packing strategy?**

Most certainly, in the case that the packing heuristic does not find the optimum solution, it finds a near optimal solution. With a high probability the optimal solution can be found with a packing heuristic (Packing Iterative especially), a simple local search or a combination of the two. The relative low computation complexity of these procedures might imply the feasibility of a two stage algorithm[12] that produces multiple tours for which it finds packing plans with a greedy packing heuristic before evaluation.

Packing Iterative seems the best packing heuristic and the uniform binary search approach of finding the best value for $\alpha$ can be justified by the 'concave' fitness function for all possible values of alpha.

The fact that the optimal packing plan can be found with relative ease raises some existential questions. How difficult is the TTP exactly if for any given tour the optimal packing plan can be found with minimal computation time and with a high reliability? This could imply that the best way to solve TTP is by searching the tours search space and evaluating a tour by finding the optimal solution. For instances with a low number of items (like the instances under investigation) it is even feasible to solve them with dynamic programming.

[12] MMAS [63] and MATLS [36] for an example.

# 9 Neighborhood Reduction Strategies

## 9.1 Introduction

The high cost of evaluation, size of the search space and the impossibility to use incremental fitness evaluation makes the use of neighborhood reduction a necessity. The local search I use has two operators that mutate the tour: INSERTION and 2-OPT. The complexity of INSERTION can already be reduced (Section 7) but 2-OPT cannot.

In some literature the neighborhood of 2-OPT is reduced by using the Delaunay triangulation, possibly motivated by the success achieved in the TSP. It is not evident these results hold for TTP. Indeed, for TTP local optima are reported with longer edges and with edges that cross each other[13]. Therefore I want to see how these neighborhood reductions perform and if there are other neighborhood reductions used for TSP that can be used for TTP. In order to answer the following research question:

**What is the effect of using different restricted neighborhoods for the 2-Opt local search?**

The reduced neighborhood for 2-OPT means that not all possible applications of the 2-opt swap are considered. Only the operations that introduce edges which are in the neighborhood of a certain node are considered.

The neighborhoods under investigation are:

- Nearest neighbor.

  Every node has 4 (NN-4), 8 (NN-8) or 16 (NN-16) neighbors. With NN-16, the neighbors of a node are the first 16 nodes that occur when sorted on the euclidean distances between nodes.

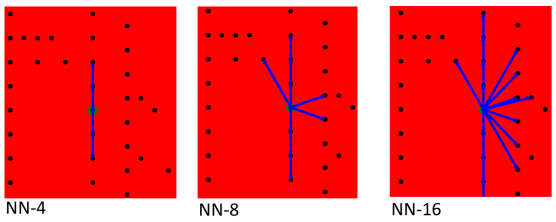


Figure 17: The edges of a node with nearest neighborhood reduction.

- k-quadrant nearest neighbor.

  Every node has 4 (QN-1), 8 (QN-2) or 16 (QN-4) neighbors. The neighbors of a node are the first $k$ nodes that occur in each quadrant when sorted on the euclidean distances between nodes.

[13] A property in euclidean TSP is that a solution can always be improved if the tour crosses itself. This does not hold for TTP.

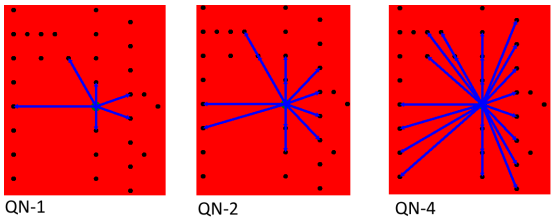


Figure 18: The edges of a node with k-quadrant nearest neighborhood reduction.

- Neighbor in a Delaunay triangulation.

  Only edges found in the Delaunay triangulation (DT) are considered. Another practice is to use neighbors of neighbors as edges (DT2).

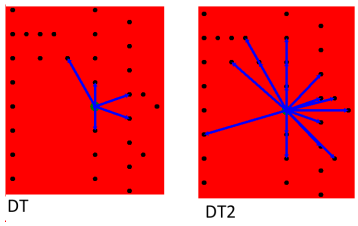


Figure 19: The edges of a node with Delaunay triangulation neighborhood reduction.

The 8 different neighborhoods all grow linear in the number of cities. Table 6 shows how many edges there are for each neighborhood for each instance. The number can be lower than you might except since duplicates are omitted for obvious reasons.

Table 6: Number of edges for each TSP-instance and each neighborhood

| | NN-4 | NN-8 | NN-16 | QN-1 | QN-2 | QN-4 | DT | DT2 | Normal |
|---|---|---|---|---|---|---|---|---|---|
| a280 | 655 | 1298 | 2526 | 749 | 1396 | 2651 | 790 | 2437 | 39060 |
| berlin52 | 140 | 280 | 564 | 130 | 243 | 447 | 145 | 426 | 1326 |
| bier127 | 340 | 672 | 1389 | 332 | 632 | 1201 | 368 | 1140 | 8001 |
| ch130 | 319 | 622 | 1235 | 347 | 653 | 1191 | 377 | 1183 | 8385 |
| eil51 | 124 | 246 | 490 | 116 | 216 | 388 | 140 | 391 | 1275 |
| kroA100 | 234 | 470 | 931 | 270 | 486 | 881 | 285 | 877 | 4950 |
| pr76 | 187 | 368 | 742 | 200 | 373 | 655 | 218 | 665 | 2850 |
| rat195 | 454 | 886 | 1747 | 487 | 916 | 1746 | 562 | 1721 | 18915 |

## 9.2 Experimental Setup

The instances I investigate are the same as in the previous chapter. The local search performed will be iterative doing a first improvement step in BitFlip,

SWAP, INSERTION and 2-OPT (with the reduced neighborhood). In total 24000 local optima will be produced. In order to comparable fitness values between instances I will map these values between 0 and 1.

## 9.3 Results

The average scores grouped by TSP-instances are shown in Table 7 and Figure 20.

It would be no surprise that the local search with no neighborhood reduction provides the best average on all instances. But at a great cost. The additional factor $O(n)$ in increased running time of the whole 2-OPT local search is noticeable with instance of $a280$[14]. Here it becomes apparent that 2-OPT is indeed the bottleneck of the local search if the neighborhood is not reduced. This will only grow more rapidly for a large amount of cities.

The difference between neighborhood reduction strategies are non-satisfactory. If you look at the three best performing neighborhoods across all instances you will see that NN-16, QN-4 and DT2 all have comparable fitness scores. On average these neighborhoods are second best in terms of ordering[15].

It seems that which neighborhood performs best depends on the structure. And even then the difference between NN-16, QN-4 and DT2 is not statistically significant. The trends are almost the same for all instances at different levels (Figure 20).

Table 7: Table containing the average fitness scores, mapped to values between 0 and 1, for each instance.

| | NN-4 | NN-8 | NN-16 | QN-1 | QN-2 | QN-4 | DT | DT2 | Normal |
|---|---|---|---|---|---|---|---|---|---|
| a280 | 0.4814 | 0.5053 | 0.5449 | 0.4828 | 0.5247 | 0.5671 | 0.4834 | 0.5564 | 0.6914 |
| berlin52 | 0.6931 | 0.7706 | 0.8181 | 0.7232 | 0.7722 | 0.8053 | 0.7258 | 0.8022 | 0.8373 |
| bier127 | 0.6498 | 0.7121 | 0.7631 | 0.6827 | 0.7268 | 0.762 | 0.6825 | 0.7600 | 0.7999 |
| ch130 | 0.4922 | 0.5325 | 0.5774 | 0.5109 | 0.5437 | 0.5837 | 0.5175 | 0.5844 | 0.6505 |
| eil51 | 0.6172 | 0.6597 | 0.6977 | 0.6224 | 0.6532 | 0.6874 | 0.6324 | 0.6879 | 0.7303 |
| kroA100 | 0.5598 | 0.6082 | 0.6477 | 0.5845 | 0.6209 | 0.648 | 0.5794 | 0.6492 | 0.6905 |
| pr76 | 0.5672 | 0.6214 | 0.6599 | 0.5884 | 0.6224 | 0.6531 | 0.5827 | 0.6543 | 0.7023 |
| rat195 | 0.5108 | 0.5349 | 0.5677 | 0.5083 | 0.5361 | 0.5845 | 0.5102 | 0.5773 | 0.6855 |
| avg | 0.5715 | 0.6181 | 0.6596 | 0.5879 | 0.6250 | 0.6614 | 0.5892 | 0.6590 | 0.7234 |

[14] Appendix A.2 (Table 25)
[15] Appendix A.2 (Table 24)

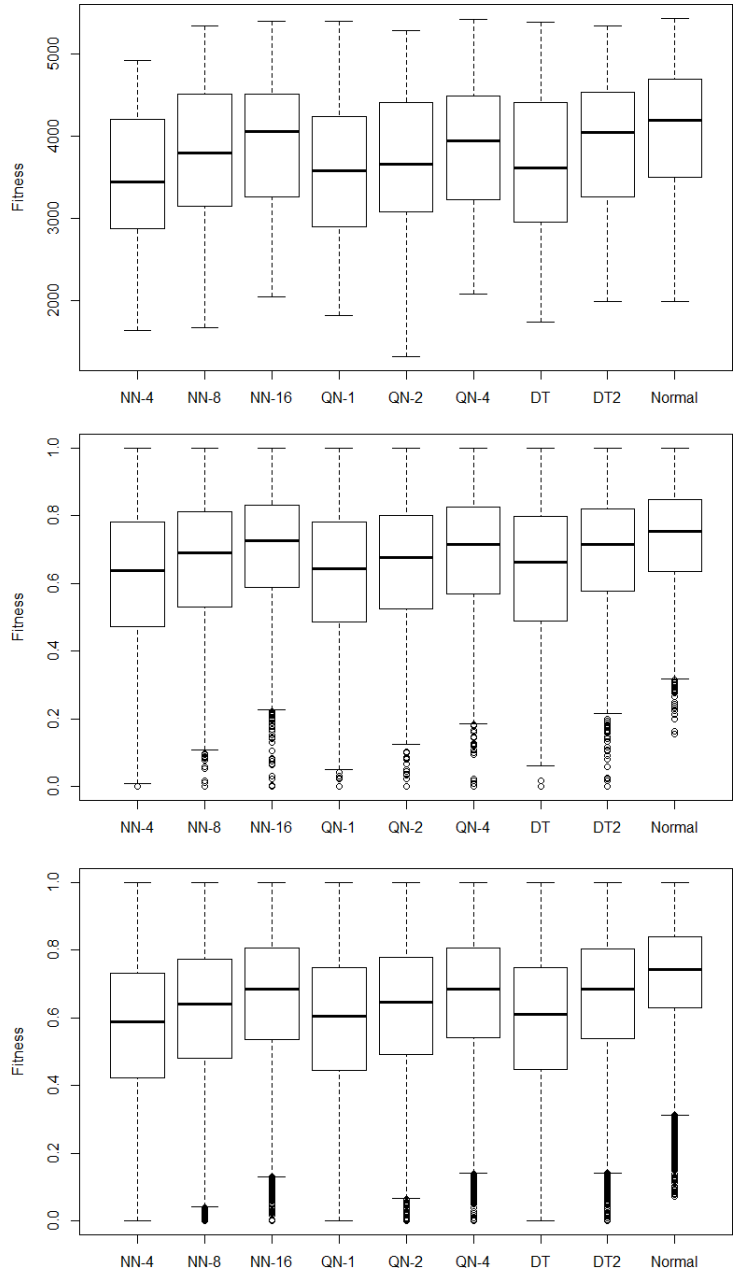


Figure 20: The top plot is a single instances of *eil*51. The middle plot are all instances of *eil*51 combined. The bottom plot are all instances combined.

# 10 Fitness Landscape Analysis

In this section I want to answer the following research question:

**What are the characteristics of the fitness landscape of various operators of traveling thief problem instances?**

In order to achieve this, I will perform the following fitness landscape analysis: autocorrelation, probability to find a local optimum (basin of attraction) and fitness distance correlation of the following operators: BitFlip, 2-opt, Exchange, Insertion and all combined in a local search procedure. The measures and visualization of the measures can give insight in what kind of metaheuristics are effective for the TTP.

## 10.1 Autocorrelation

The autocorrelation can give insight in the ruggedness of a landscape. In general the lower the autocorrelation the higher the ruggedness of a landscape. In order to give perspective to these measures I will compare the autocorrelation of the operators under investigation in the context of their original problem; The autocorrelation of 2-opt and Insertion in a TSP instance compared to a similar TTP instance. Same for BitFlip and Exchange with the knapsack problem.

To approximate the autocorrelation I will use a random walk of 100.000 applications of the operator under investigation. This gives an approximation of the autocorrelation:

$$R(1) \approx \frac{1}{\sigma_f^2(n-1)} \sum_{i=1}^{n-1} (f(x_i) - \overline{f})(f(x_{i+1}) - \overline{f})$$

I studied the same instances as in the previous chapters (240 instances in total). This gives an insight in how the autocorrelation changes when knapsack type, capacity category, TSP instance and item factor changes.

### 10.1.1 Results

From the results it becomes apparent that the ruggedness of 2-opt is much larger than Insertion (Figure 21). This difference can be explained by the fact that 2-opt for TTP is much more disruptive than Insertion. More disruptive since the order of cities is of importance. Insertion only affects the ordering of one element while 2-opt swaps the ordering of a whole segment.

This is different for the TSP where the ordering does not matter intrinsically. 2-opt for TSP only affects 4 edges. Those that are removed and those that are added. Therefore the autocorrelation of 2-opt is much higher for TSP than for TTP[16].

Figure 21 shows an interesting trend. The change in autocorrelations suggest that the larger the capacity of the knapsack is the more rugged the landscape

[16] Appendix A.3 (Figure 38)

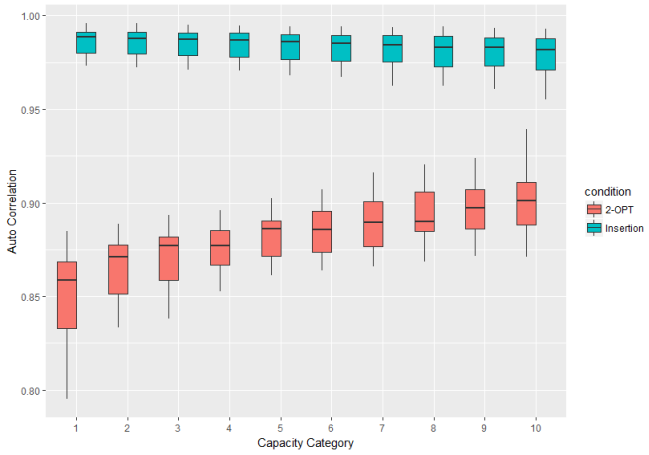


Figure 21: Boxplot of the autocorrelation of 2-OPT and INSERTION for all instances grouped by different capacity categories.

of 2-OPT becomes. This phenomena could be explained the following way: 2-OPT is disruptive if the picked up items are distributed over the segment in an irregular manner. The larger the capacity, the more items are picked up and the higher the chance that a segment has a regular distribution of item weight.

The slight drop of the autocorrelation for INSERTION can be explained by the fact that a city with an item that is inserted is more disruptive than if the city did not contain an item.

A similar trend can be seen for the item factor. The higher the item factor the more rugged the landscape of 2-OPT becomes and the less rugged the landscape ofINSERTION becomes.[17]

The autocorrelation for 2-OPT and INSERTION increases when the amount of cities increases. This is a normal property of the autocorrelation measure. The bigger the solution representation the lower the effect on the fitness value for small changes on that solution representation.

The autocorrelation of 2-OPT and INSERTION for the different knapsack types does not differ that much (Figure 23). The only noticeable difference is the spread for 2-OPT of the type uncorrelated with similar weights. It is higher compared to the other types. This is due to the interaction between the constant weight of every item and the capacity constraint. The lowest knapsack capacity with the type uncorrelated with similar weights has the lowest autocorrelation for every TSP-instance compared to other types and knapsack capacities. Similar for the highest knapsack capacity that has the highest autocorrelation compared to others.

For the knapsack operators similar results can be seen. The autocorrelation increases when the knapsack capacity increases (Figure 24). Surprisingly the

[17]Appendix A.3 (Figure 40)

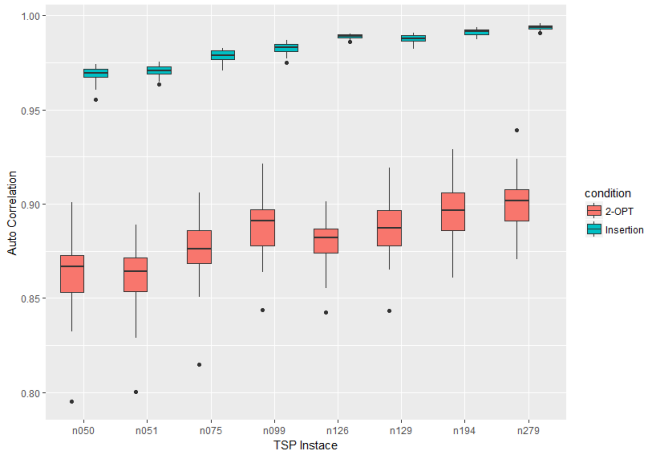


Figure 22: Boxplot of the autocorrelation of 2-OPT and INSERTION for all instances grouped by TSP-instances.

differences between the normal knapsack problem and the knapsack problem as a subproblem of the TTP are minimal. For some instances the autocorrelation is higher than for the normal knapsack problem[18] .

[18] the outliers of bounded strongly correlated (Appendix A.3 Figure 41)

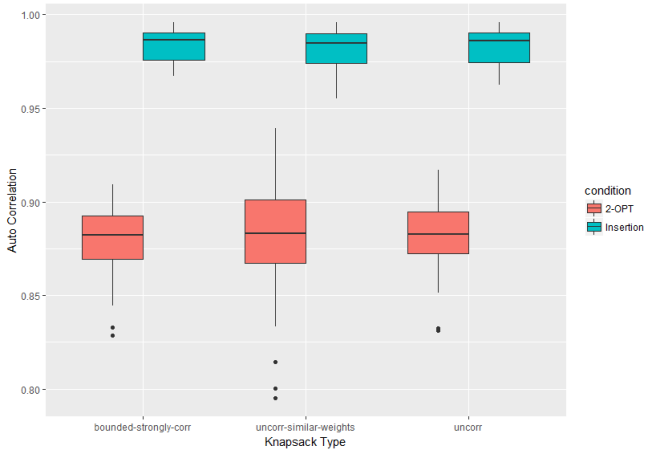


Figure 23: Boxplot of the autocorrelation of 2-OPT and INSERTION for all instances grouped by different knapsack types

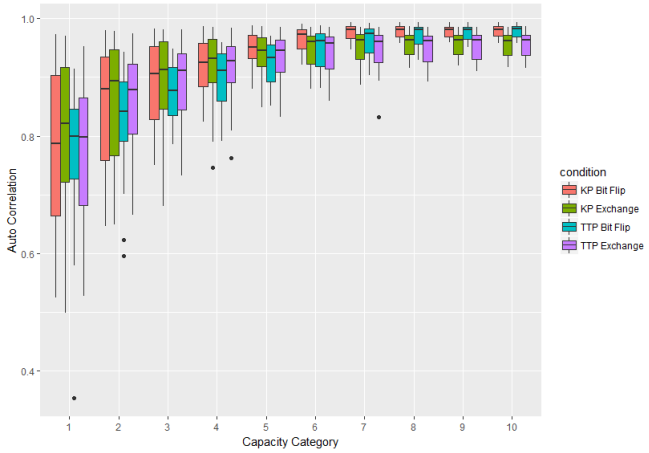


Figure 24: Boxplot of the autocorrelation of FLIP and EXCHANGE for all instances grouped by different capacity categories.

## 10.2 Size of the Region of Attraction

In order to give insight in how many different local optima there exist and how high the probability is to find such a local optima I have produced 1.000.000 local optima with the normal local search procedure (without neighborhood reduction) and counted the number of times the same local optimum has been found. I have done this for 9 instances of $eil51$ with 3 different capacity categories $\in \{01, 05, 10\}$ and 3 different knapsack types.

### 10.2.1 Results

Surprisingly, for all instances the percentage of found local optima that are unique is incredibly high (Table 8). This ranges between 19.5% and 80%. The average number occurrences therefore is pretty low but there are some optima which are found more often (Figure 25). Unfortunately there seems to be no correlation between fitness and frequency of optima found. But the local optima that are found more often seem to have a high fitness. But having a high fitness does not mean the optimum is found more often. In only one of the instances *USW01* the optimum that is found most often is also the best found.

Table 8: Results of the 1000000 local optima for the different instances

| | Capacity | Unique optima | Avg frequency | Max frequency | Frequency of best found |
|---|---|---|---|---|---|
| BSC | | | | | |
| | 01 | 683744 | 1.46 | 323 | 47 |
| | 05 | 798575 | 1.25 | 297 | 59 |
| | 10 | 443050 | 2.26 | 1257 | 32 |
| USW | | | | | |
| | 01 | 516804 | 1.93 | 708 | 708 |
| | 05 | 592795 | 1.69 | 550 | 268 |
| | 10 | 200905 | 4.98 | 4446 | 898 |
| U | | | | | |
| | 01 | 358996 | 2.79 | 1810 | 45 |
| | 05 | 348335 | 2.87 | 4127 | 3110 |
| | 10 | 195249 | 5.12 | 2181 | 362 |

The fact that there are many unique local optima and that the probability of finding the global optimum is very low implies that using multistart local search is not the best strategy.

The total number of unique packing plans varies for different instances. Both uncorrelated instances with capacity category 01 have a relative low number of unique packing plans and for the instance *USW01* in 65% of the cases the same packing plan is found. This can be due to the fact that instance *USW01* has a knapsack capacity of 4567 while all items have a weight between 1000 and 1010. Therefore there is only room for 4 different items in the knapsack. Combined with the fact that the ratios are uncorrelated this makes some item preferable over others. This can explain why the same packing plan is found many times.

On the other hand the amount of unique packing plans for other instances is pretty high (45% for the instance *BSC01*).

Table 9: Results of the 1000000 local optima for the different instances of the different found packing plans.

| | capacity | unique plans | avg frequency | max frequency |
|---|---|---|---|---|
| BSC | | | | |
| | 01 | 20520 | 48.73 | 20964 |
| | 05 | 454387 | 2.2 | 2239 |
| | 10 | 156048 | 6.41 | 6918 |
| USW | | | | |
| | 01 | 2084 | 479.85 | 64915 |
| | 05 | 119074 | 8.4 | 9848 |
| | 10 | 12421 | 80.51 | 38264 |
| U | | | | |
| | 01 | 1285 | 778.21 | 146594 |
| | 05 | 23416 | 42.71 | 17038 |
| | 10 | 7674 | 130.31 | 44497 |

Same as for the whole problem there is a high percentage of unique tours almost equaling the amount of unique local optima (Table 10 & Figure 27). For the instance *U01* the high amount of unique local optima is almost entirely due to the high amount of unique tours.

Table 10: Results of the 1000000 local optima for the different instances of the different found tours.

| | capacity | unique tours | avg frequency | max frequency |
|---|---|---|---|---|
| BSC | | | | |
| | 01 | 649288 | 1.54 | 324 |
| | 05 | 703729 | 1.42 | 372 |
| | 10 | 377633 | 2.65 | 1259 |
| USW | | | | |
| | 01 | 514221 | 1.94 | 708 |
| | 05 | 591560 | 1.69 | 550 |
| | 10 | 200766 | 4.98 | 4446 |
| U | | | | |
| | 01 | 351367 | 2.85 | 1810 |
| | 05 | 344713 | 2.9 | 4127 |
| | 10 | 194221 | 5.15 | 2181 |

One remarkable thing is seen in Figure 25-27. For the instance *BSC01* there is a cluster of local optima with a relatively low fitness. The cluster of $\sim$2500 local optima all share the same packing plan but differ in their tours. Interestingly there exists a packing plan with a high region of attraction but with a significantly lower fitness than all other optima. It would be interesting to see (next chapter) if an iterated local search can escape out of this cluster of local optima.

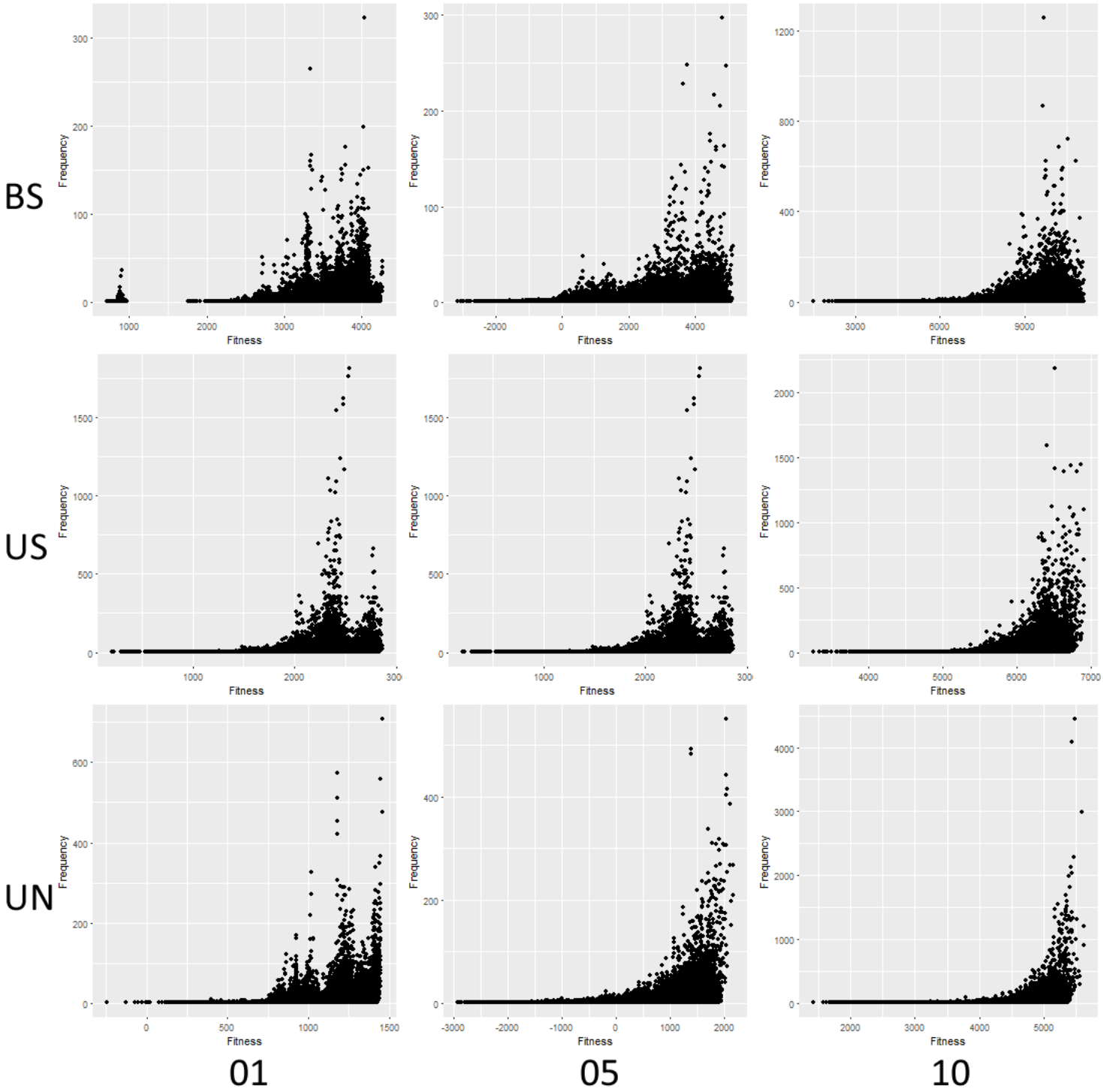


Figure 25: Plot of the 1000000 found local optima

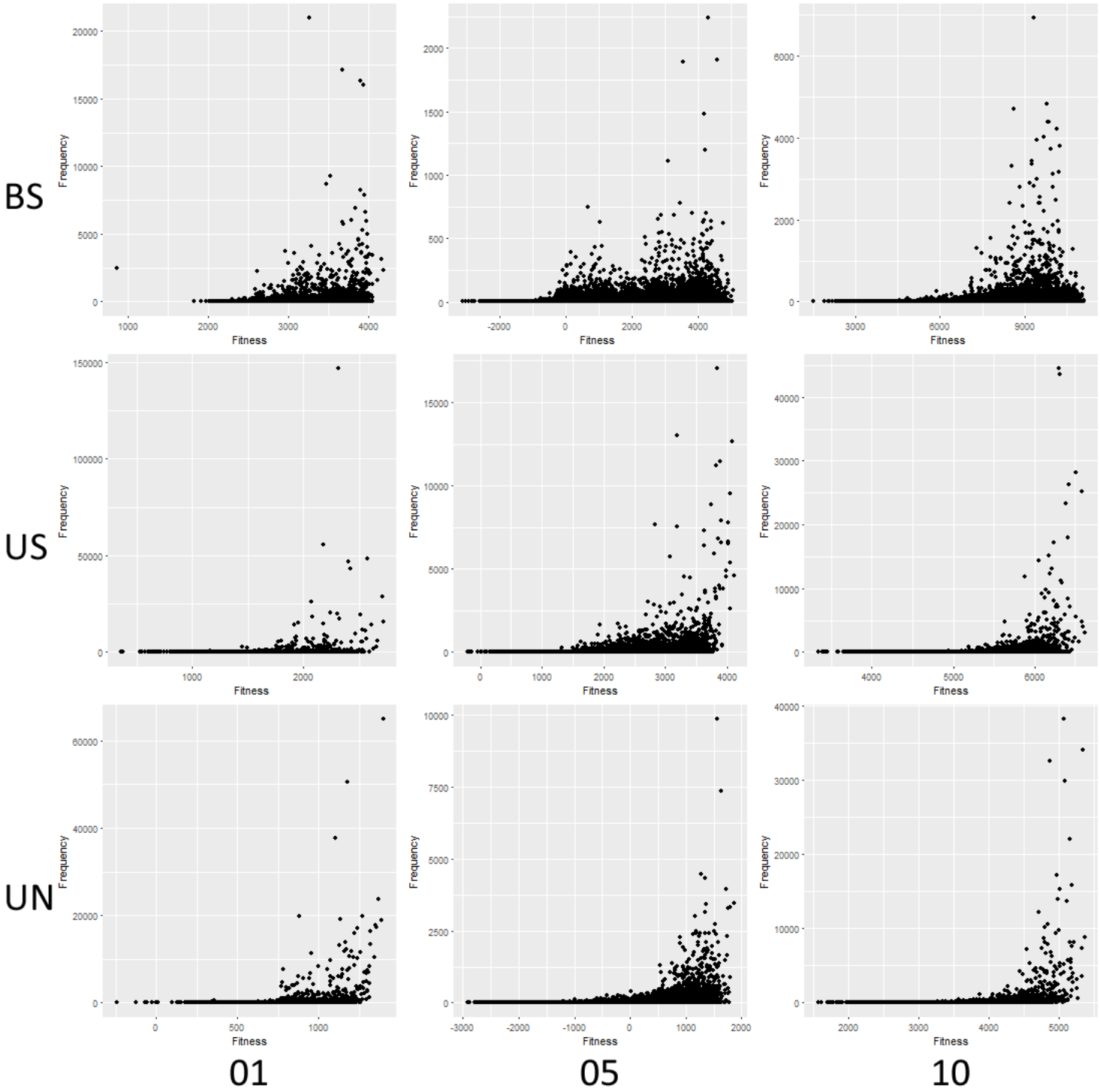


Figure 26: Plot of the 1000000 found packing plans

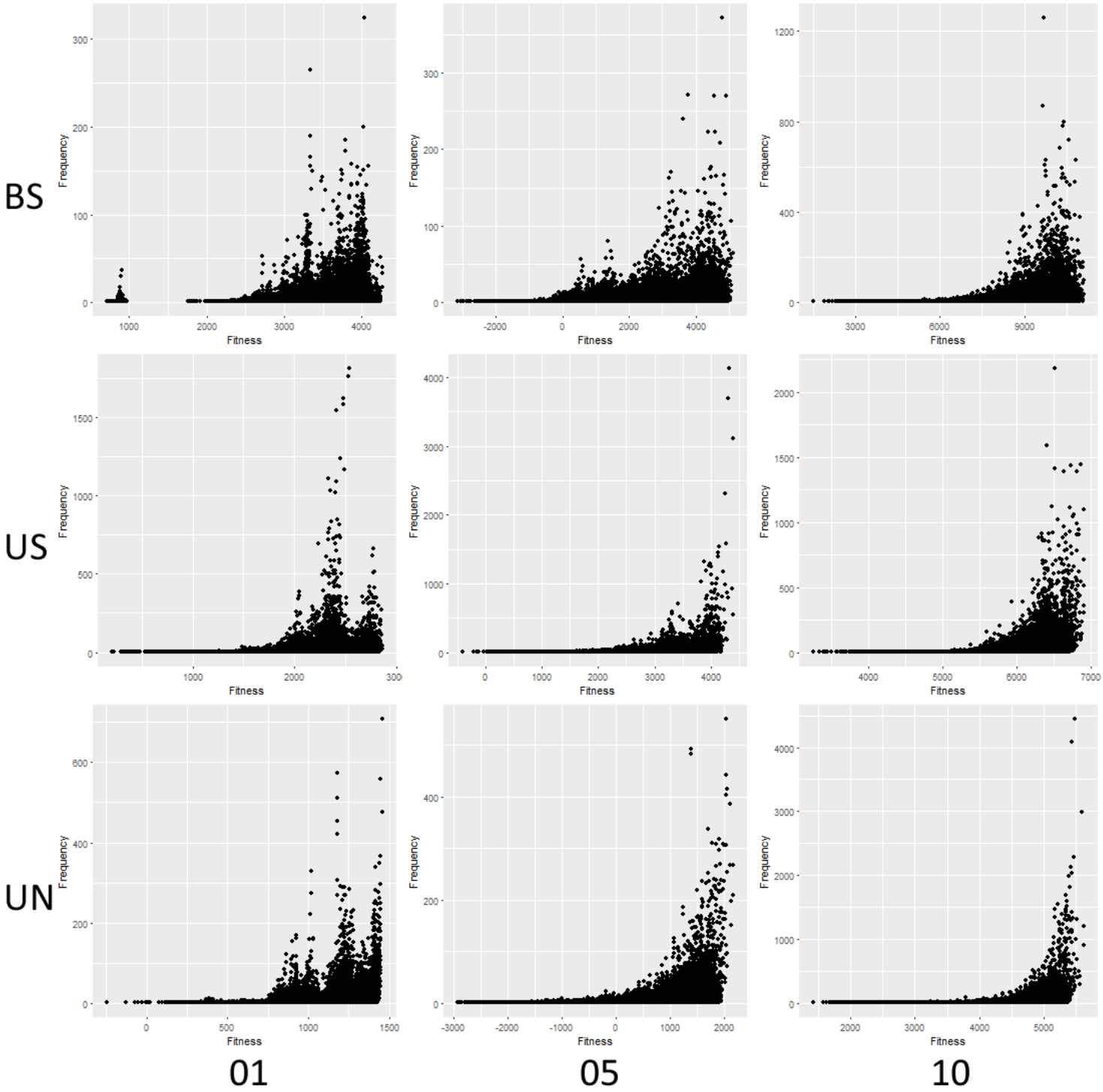


Figure 27: Plot of the 1000000 found tours

## 10.3 Fitness Distance Correlation

The fitness distance correlation (FDC) can give insight in how local optima are related to each other. A high negative correlation of fitness and distance can mean that the landscape of local optima can be easily traversed with an iterated local search.

I will use the local optima produced in the last section as data and study the same 9 instances. The FDC only has meaning when the global optimum is known. Since there is no timely manner in which the global optimum can be proven, I will make the assumption that the best found optimum out of 1.000.000 is the global optimum.

The distance between packing plans will be the hamming distance and the distance between tours will be the bond distance [5], the number of edges between tours that are not shared.

### 10.3.1 Results

Table 11 and Figure 28-30 show promising results. There is a high amount of fitness distance correlation. The FDC for every instance is $< -0.15$. According to Jones and Forrest [25] this means the landscape is straightforward. From the figures it becomes apparent that the closer you are to the best solution the higher the range of lowest and highest found optima becomes.

The same cluster of local optima with low fitness of *BSC01* is again seen in the figures.

Surprisingly there is a high correlation between the distance in the packing plans and the fitness. The highest is the instance *U05* with a correlation of $-0.70$. This implies that a packing plan includes crucial information regarding where the solution is on the landscape and where to proceed next. For the just mentioned instance it might be better to perturb the solution when doing an iterated local search in such a way that the packing plan remains mostly intact.

For every instance I have performed additional tests on the best found solution while keeping one of the subproblems fixed. i.e. finding the optimal packing plan when the tour is fixed and finding the best tour when the packing plan is fixed. Results are in the last column of Table 11. The fitness distance correlation for the tour improves tremendously and the FDC for the packing plans are even higher. For the uncorrelated instances the optimal solution is found every time when the tour is fixed. This is coherent with Section 8 which suggests that finding the optimal packing plan is relatively easy.

## 10.4 Conclusion Fitness Landscape Analysis

The fitness analysis of some instances of the traveling thief problem have given us some insights in the problem and how to solve it. Some key points:

- Insertion has a higher autocorrelation than 2-opt for the TTP. While 2-opt has has a higher autocorrelation for TSP. This suggests that Insertion is a good local search for TTP.

Table 11: Fitness Distance correlation for 9 instances of *eil*51 and 3 different distances (middle column) and additional test where one subproblem is fixed (last column). The cells that contains '-' produce the same results after every local search.

| | | 1000000 local optima | | | Fixed subproblem | |
|---|---|---|---|---|---|---|
| | Capacity | Packing plan | Tour | Combined | Packing plan | Tour |
| BSC | | | | | | |
| | 01 | -0.36 | -0.15 | -0.26 | -0.91 | -0.60 |
| | 05 | -0.55 | -0.27 | -0.58 | -0.82 | -0.69 |
| | 10 | -0.28 | -0.37 | -0.42 | -0.88 | -0.56 |
| USW | | | | | | |
| | 01 | -0.68 | -0.34 | -0.54 | - | -0.65 |
| | 05 | -0.61 | -0.47 | -0.67 | - | -0.70 |
| | 10 | -0.52 | -0.45 | -0.60 | - | -0.50 |
| U | | | | | | |
| | 01 | -0.57 | -0.31 | -0.50 | - | -0.54 |
| | 05 | -0.70 | -0.34 | -0.58 | - | -0.59 |
| | 10 | -0.48 | -0.48 | -0.61 | - | -0.55 |

- There are many unique local optima and the global optimum has a small region of attraction. This suggests that multistart local search is not a good approach.
- The fitness distance correlation is high when the distance of tour and packing plans are combined. This is even higher when one of the subproblems is kept fixed. This indicates that an iterated local search has a high chance of finding good solutions.
- The fitness distance correlation for the packing plans is high. This is interesting in combination with previous results in (Section 8) that suggested that finding the optimal packing plan for a given tour is relatively easy. But the correlation between fitness and hamming distance of packing plans suggests that you might lose valuable information when you only consider the tours and not the packing plans.

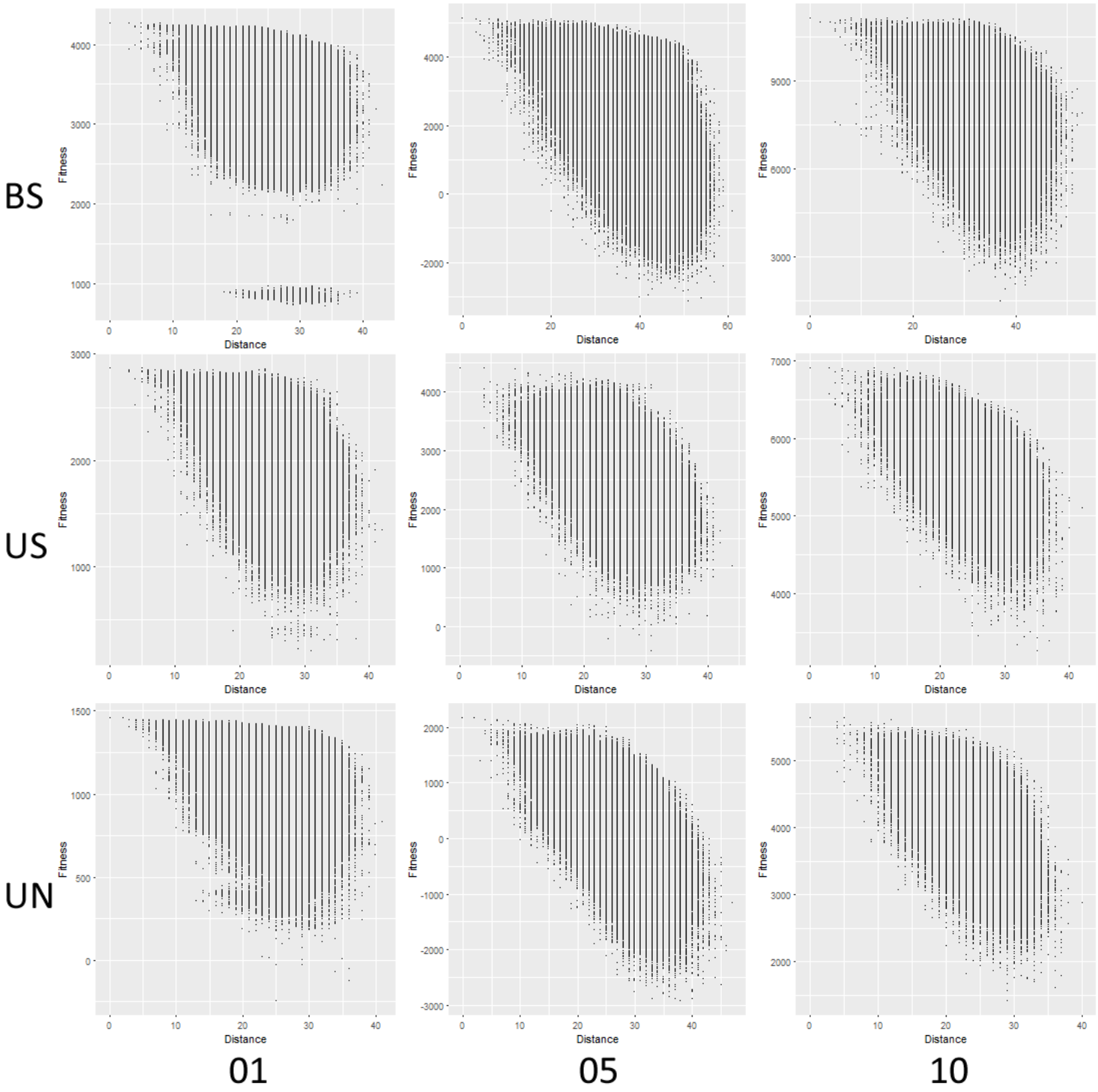


Figure 28: Plot of 1000000 local optima. The distance is the hamming distance plus the bond distance.

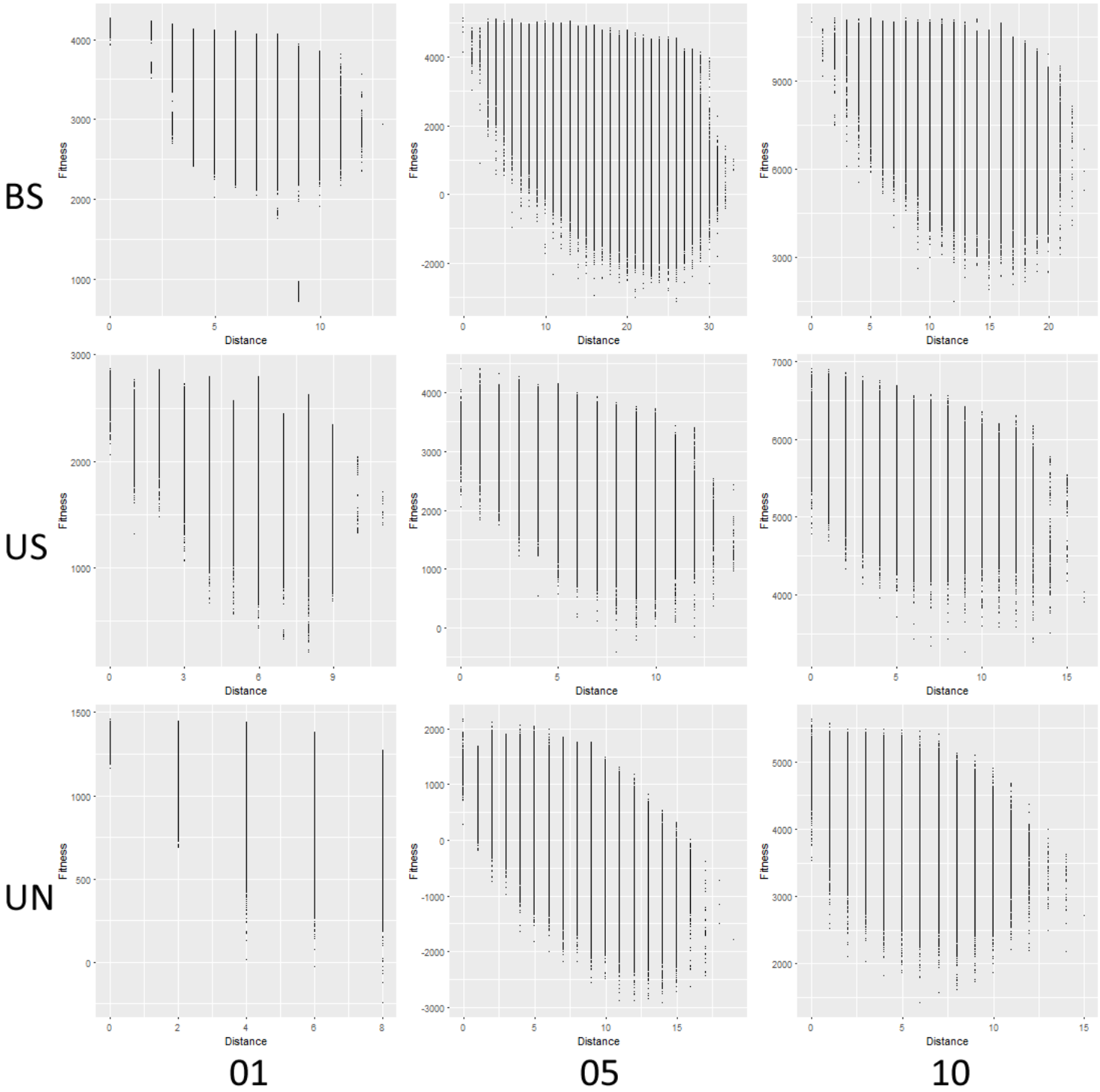


Figure 29: Plot of 1000000 local optima. The distance is the hamming distance.

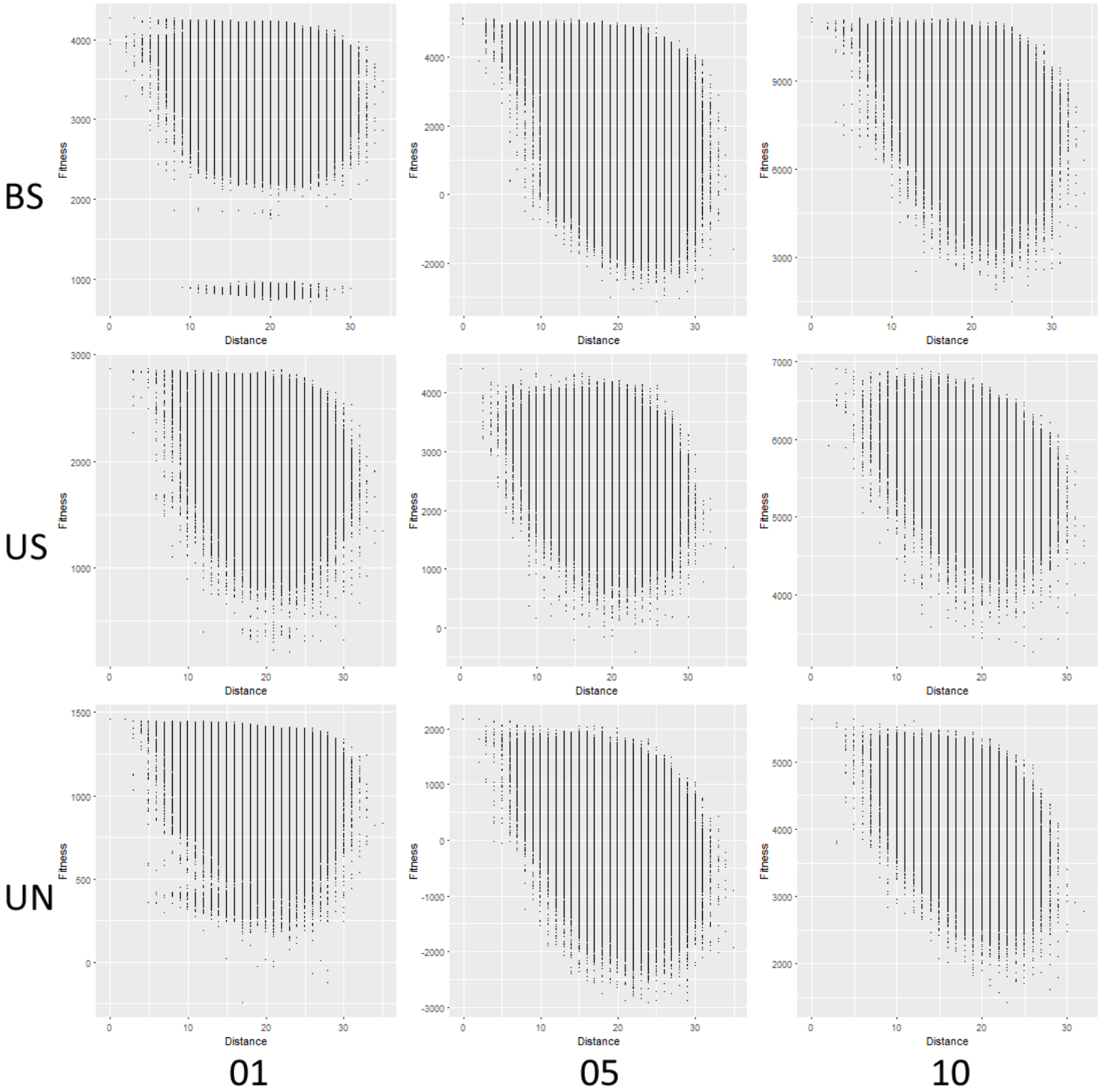


Figure 30: Plot of 1000000 local optima. The distance is the bond distance.

# 11 Iterated Local Search & Its Fitness landscape

In this section I will investigate how local optima are related to each other and find out whether TTP contains a big valley or if it has a multiple funnel structure. In order to do this I will collect local optima found with an iterated local search (ILS) and research how they are related to each other. I will answer the following research question:

**Does TTP instances have a single (big valley[19]) or multiple funnel structure?**

In this section I will also study the effect of iterated local search and how often it can find the global optimum[20].

## 11.1 Iterated Local Search

### 11.1.1 Experimental Setup

In order to answer the research question I will first investigate what a good perturbation and perturbation size is.

In preliminary experiments I found that perturbing only the tour or only the packing plan only had a small chance of kicking the solution out of its region of attraction. Therefore I investigated multiple perturbation sizes where each perturbation is a combination of $k$ random applications of the 2-opt swap and $k$ random bit flips. I ran experiments with perturbation size $\in \{1, 2, 3, 4, 5, 10, 20\}$ on the 9 instances of $eil51$ similar to those studied in the previous chapters. The ILS will stop after 2500 iterations without any improvement. This is a rather large number (for this instance) but in practice it uses far less iterations. For every instance and perturbation size the algorithm is ran 100 times. Algorithm 7 shows the basic flow of the iterated local search in pseudocode.

**Algorithm 7** Iterated Local Search

```
1:  function ILS
2:      x ← an initial solution
3:      while k ≤ 2500 do
4:          x' ← Perturb(x)
5:          if f(x) ≤ f(x') then          ▷ Where f is the fitness function
6:              x ← x'
7:              k ← 0
8:          end if
9:          k ← k + 1
10:     end while
11: end function
```

[19] big valley might not be the correct term for a maximization problem but big mountain sounds silly.

[20] We again assume the best found solution is the global optimum

### 11.1.2 Results

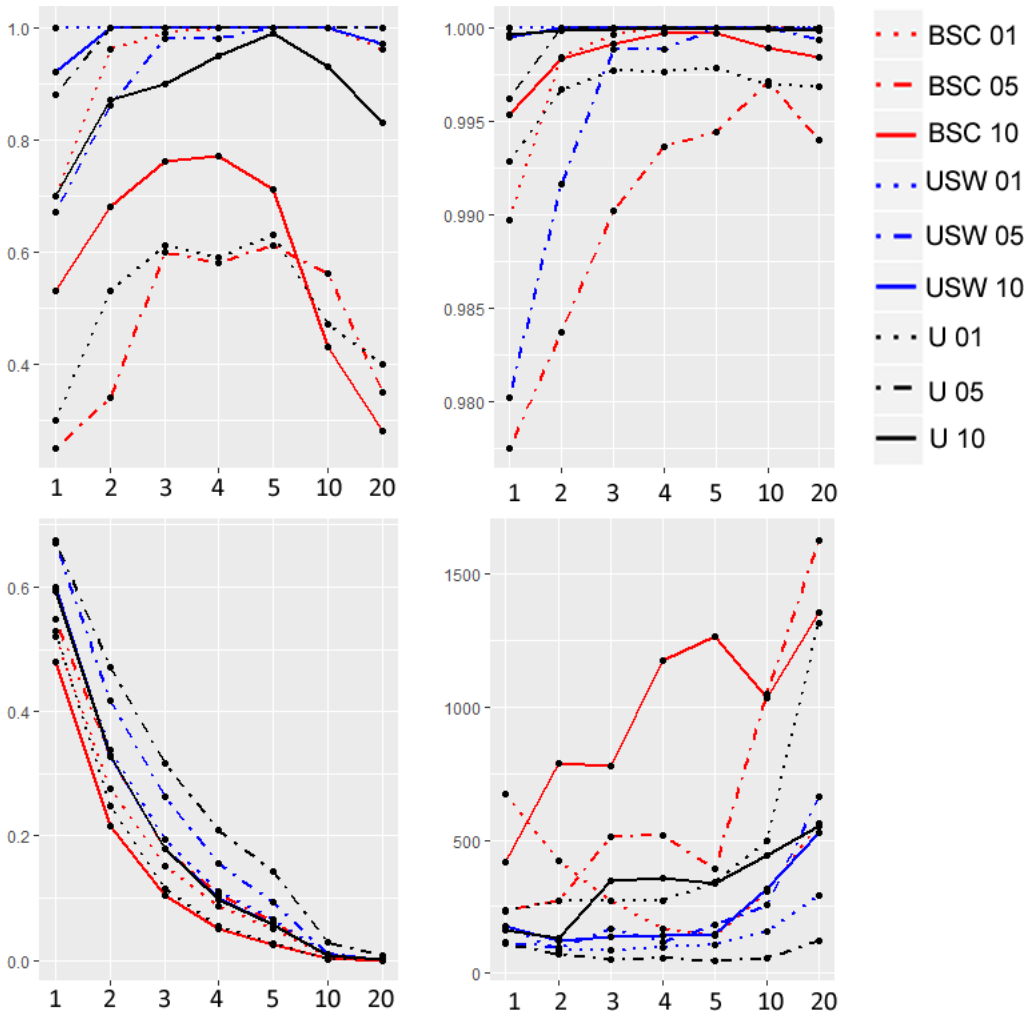


Figure 31: Probability the global optimum is found after running ILS (top left). Average fitness approximation (top right). Probability the perturbation escapes the region of attraction (bottom left). Median of the amount local optima produced before the final solution was found (bottom right).

Figure 31 and Table 12 13 14 contain the results of the experiments. For some perturbation sizes (3, 4 and 5) the chance of finding the global optimum is high. For the perturbation size of 5 and 10 (Table 12) the global optimum for all the instances of type *USW* is found 100 times out of a 100 runs. For the *BSC* instances the effectiveness of iterated local search versus multistart local search becomes apparent. When the perturbation acts like a random restart the global

optimum is only found in 12% or 17% of the time.

A perturbation size of 5 is the best size in terms of the probability of finding the global optimum. It is also the best size for the best fitness approximation on the instances studied.

The difference between multistart local search and ILS would be even greater if the maximum number of non-improving is lower. Figure 31 and Table 14 in Appendix A.4 show that indeed the maximum number of non-improving could be lowered.

On average the largest perturbation size also uses the most perturbations before halting, this is because with such a large perturbation size it almost becomes a random restart.

Figure 31 shows that the probability of finding the global optimum and fitness approximation increases when the perturbation size becomes higher. It peaks at 5 and then falls off with larger perturbation sizes of 10 or 20. Surprisingly even using relatively high perturbation sizes give better results than doing a random restart (Table 12 & Table 13). This means that there is still some structure preserved in the perturbed solutions.

Table 14 shows the probability of return to the same local optimum. This is around 60% for the smallest and almost zero for the largest perturbation size. The perturbation size of 5, which gave the best results, returns between 2.5% and 15%. This means that the solution is changed enough that it almost always falls outside the region of attraction but is also similar enough to return the same local optima sometimes. Therefore this perturbation does not stray too far away from the initial solution.

Table 12: Probability the global optimum was found after running ILS

| perturbation size | | 1 | 2 | 3 | 4 | 5 | 10 | 20 | Random |
|---|---|---|---|---|---|---|---|---|---|
| BSC | | | | | | | | | |
| | 01 | 0.7 | 0.96 | 0.99 | 1 | 1 | 1 | 0.96 | 0.17 |
| | 05 | 0.25 | 0.34 | 0.6 | 0.58 | 0.61 | 0.56 | 0.35 | 0.17 |
| | 10 | 0.53 | 0.68 | 0.76 | 0.77 | 0.71 | 0.43 | 0.28 | 0.12 |
| USW | | | | | | | | | |
| | 01 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0.97 |
| | 05 | 0.67 | 0.86 | 0.98 | 0.98 | 1 | 1 | 0.97 | 0.85 |
| | 10 | 0.92 | 1 | 1 | 1 | 1 | 1 | 0.97 | 0.95 |
| U | | | | | | | | | |
| | 01 | 0.3 | 0.53 | 0.61 | 0.59 | 0.63 | 0.47 | 0.4 | 0.27 |
| | 05 | 0.88 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 10 | 0.7 | 0.87 | 0.9 | 0.95 | 0.99 | 0.93 | 0.83 | 0.92 |

Table 13: Average fitness approximation.

| | | 1 | 2 | 3 | 4 | 5 | 10 | 20 | Random |
|---|---|---|---|---|---|---|---|---|---|
| BSC | | | | | | | | | |
| | 01 | 0.9897 | 0.9984 | 0.9996 | 1 | 1 | 1 | 1 | 0.9952 |
| | 05 | 0.9775 | 0.9837 | 0.9902 | 0.9936 | 0.9944 | 0.9971 | 0.994 | 0.9859 |
| | 10 | 0.9953 | 0.9983 | 0.9991 | 0.9997 | 0.9997 | 0.9989 | 0.9984 | 0.9959 |
| USW | | | | | | | | | |
| | 01 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0.9998 |
| | 05 | 0.9802 | 0.9916 | 0.9988 | 0.9988 | 1 | 1 | 0.9993 | 0.997 |
| | 10 | 0.9995 | 1 | 1 | 1 | 1 | 1 | 0.9999 | 0.9999 |
| U | | | | | | | | | |
| | 01 | 0.9928 | 0.9967 | 0.9977 | 0.9976 | 0.9978 | 0.9969 | 0.9968 | 0.9955 |
| | 05 | 0.9962 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 10 | 0.9996 | 0.9998 | 0.9999 | 0.9999 | 1 | 0.9999 | 0.9998 | 0.9999 |

Table 14: Probability the perturbation escapes the region of attraction.

| | | 1 | 2 | 3 | 4 | 5 | 10 | 20 |
|---|---|---|---|---|---|---|---|---|
| BSC | | | | | | | | |
| | 01 | 0.5201 | 0.2751 | 0.1514 | 0.0863 | 0.0513 | 0.0067 | 0.0009 |
| | 05 | 0.5479 | 0.3275 | 0.1796 | 0.1073 | 0.0635 | 0.0056 | 0.0003 |
| | 10 | 0.4804 | 0.2166 | 0.1045 | 0.0516 | 0.0258 | 0.0027 | 0.0004 |
| USW | | | | | | | | |
| | 01 | 0.5942 | 0.339 | 0.194 | 0.1109 | 0.0653 | 0.0093 | 0.0025 |
| | 05 | 0.6713 | 0.4187 | 0.2639 | 0.1568 | 0.094 | 0.0097 | 0.0016 |
| | 10 | 0.5989 | 0.3281 | 0.1796 | 0.1003 | 0.0563 | 0.0077 | 0.0018 |
| U | | | | | | | | |
| | 01 | 0.5295 | 0.2479 | 0.1151 | 0.0555 | 0.0283 | 0.0025 | 0.0005 |
| | 05 | 0.676 | 0.4712 | 0.3168 | 0.2101 | 0.1423 | 0.0286 | 0.0082 |
| | 10 | 0.5941 | 0.3288 | 0.1794 | 0.0987 | 0.0566 | 0.0069 | 0.0018 |

## 11.2 Fitness Landscape of ILS

By taking the perturbation size of 5 we can investigate how successful ILS is and what the fitness landscape of local optima looks like. I will follow a similar approach to Ochoa and Veerapen [46] who've looked at the landscape of local optima in the context of an iterated local search. They made a local optima network where local optima are nodes and edges are successful perturbations (i.e. perturbations that led to a better local optima). Their resulting graph showed that TSP contained a multi-funnel landscape with local optima which were non-optimal and for which the CLK could not escape.

I will do the same for the TTP. The procedure is elaborated in Algorithm 8[21].

[21]Similar to Ochoa and Veerapen [47, 46] but the sample size is not based on the input size but on another termination condition, 2500 non-improving perturbations.

**Algorithm 8** Local optima network sampling

```
1: function ILS
2:     x ← an initial solution
3:     while k ≤ 2500 do
4:         x' ← Perturb(x)
5:         if f(x) ≤ f(x') then
6:             N ← N ∪ {x, x'}                ▷ Store optima as node
7:             E ← E ∪ {(x, x')}              ▷ Store successful perturbation as edge
8:             k ← 0
9:         end if
10:    end while
11: end function
```

### 11.2.1 Results

Table 15: Solutions found after running ILS 100 times. The fitness values with a star corresponds to local optima where there is chance that the ILS escapes it.

| | BSC | USW | U |
|---|---|---|---|
| 01 | 4269.36 (100) | 1459.95 (42) | 2871.36 (63) |
| | | 1459.95 (58) | 2854.54 (37)* |
| 05 | 5138.39 (61) | 2171.99 (51) | 4407.56 (100) |
| | 5104 (30)* | 2171.99 (49) | |
| | 4992.29 (1)* | | |
| | 4925.42 (8)* | | |
| 10 | 11136.13 (71) | 5631.52 (100) | 6905.74 (52) |
| | 11132.59 (21)* | | 6905.74 (47) |
| | 11110.14 (8)* | | 6897.34 (1)* |

Table 15 shows all local optima found by the ILS in 100 runs. The produced graphs are displayed in Figure 32. For some instances the optimum is always found (*BSC01*, *USW01*, *USW01*, *USW01* and *U05*). Note that there are some instances with global optima that have the same fitness scores. These global optima lie close to each other and their only difference is in their tour, they all have the same packing plan. Tours can easily be equivalent when no items are picked up in a tour segment since the instance *eil_51* has low integer distances.

The landscapes of local optima for the instances of *BSC01* and *USW01* are straightforward. The landscape looks like one big funnel with the global optimum at the end. Along the way there are no local optima that get visited significantly more. This is different for *USW05*, *U05* and especially *USW10*. For *USW10* there are two other optima that have a high region of attraction (see Figure 25) and are visited a large number of times. But in all those times it eventually escapes its region of attraction and goes to the global optimum.

For the other instances an alternative local optima is found as endpoint of the ILS with a high chance (*BSC05*, *BSC10* and *U01*). Note however that for

all endpoints of the ILS that are not optimal there does exist a perturbation that escapes from it (see Figure32 it holds that: x is a global optimum $\Longleftrightarrow$ x has no downward path). This is due to the lenient termination criteria combined with a large perturbation size.

The non-global-optima endpoints are indicated in Figure 32. From the picture it becomes apparent that the alternative local optima of $U01$ is very far away from the global optimum and has many edges leading to it. They are so far apart that in 37 out of a 100 runs it could not escape it even with the relatively large perturbation size.

Instance *BSC05* has found the global optimum the least amount of times. To investigate this instances further, I ran additional tests. I have ran the ILS 1000 times but now with different termination criteria: the ILS stops after 1000 non improvement moves. The results are displayed in Figure 33. For these instances and for these ILS parameters there seems to exist a multi-funnel structure. In the bottom right corner it becomes apparent one of the non-optimal endpoints is far removed for other local optima. There exists a large gap.

## 11.3 Conclusions

ILS can produce good results, better than MLS. A perturbation size of 5 random applications of 2-opt swap and 5 random bit flips seems to perform best. For some instances the global optimum is easily found and for some of these the fitness landscape of local optima has a single funnel structure. For those instances the fitness landscape is globally concave and has a big valley structure.

For other instances finding the global optimum is harder but is still done in 50% of the cases. Some of these instances show a multi funnel structure. And this is only for the smallest instances ($eil\_51$) I suspect this is even more so for the bigger instances since they have even more local optima.

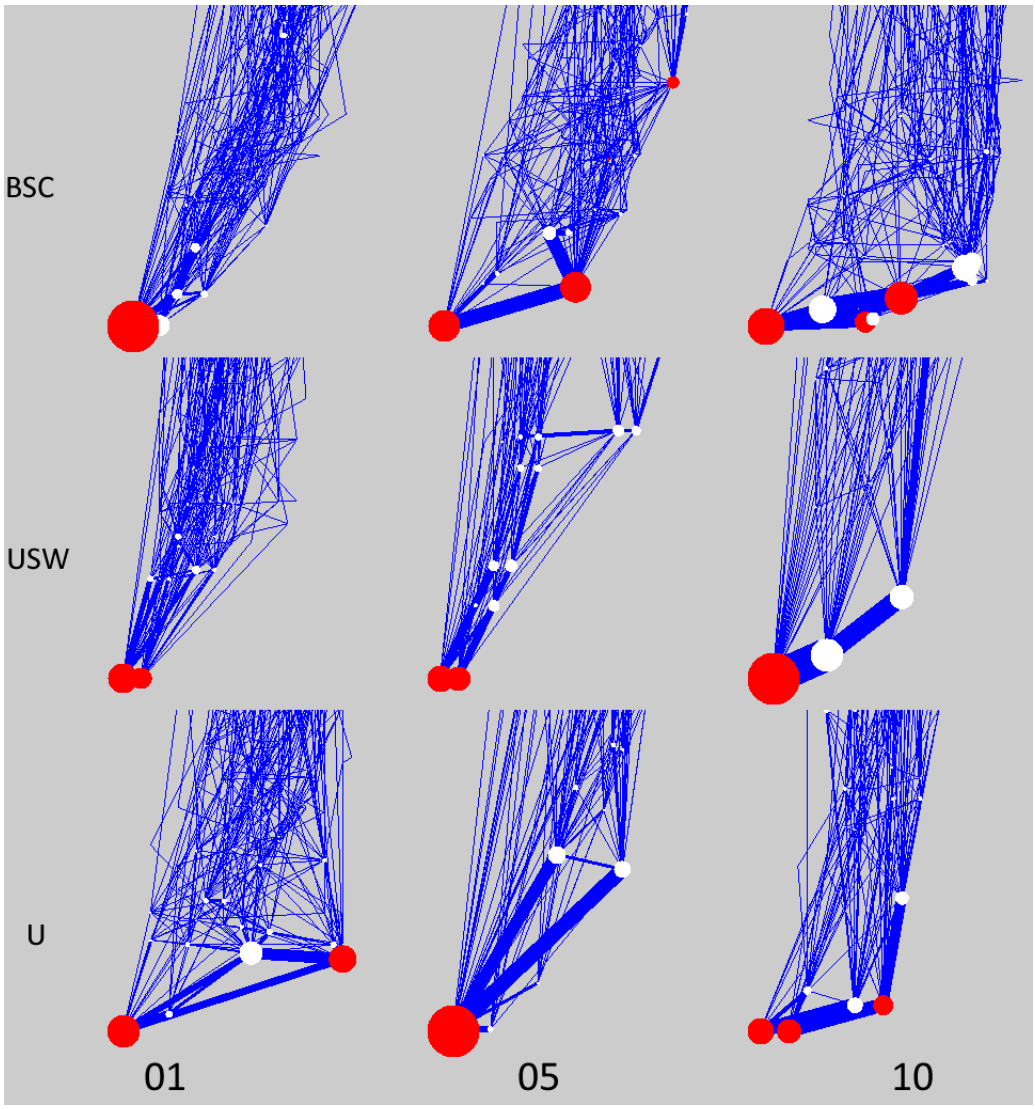


Figure 32: A plot of multiple runs of ILS. The nodes are local optima and the edges are successful perturbations. The x-axis corresponds to distance removed from the global optimum (origin at bottom left corner) and the y-axis corresponds to fitness (the lower the better). The thickness is the weight of the edge, the amount of times it occurs. The radius of a node is the amount of times that node is visited. The red nodes are endpoints of the ILS.

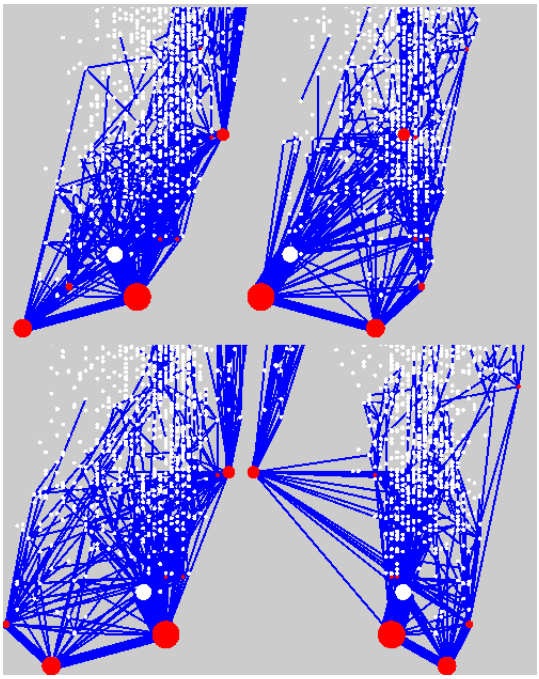

Figure 33: Graph representing 1000 runs of ILS on *bounded-strongly-corr_05*. The 4 plots are the same graph but with different x-axis. In each one an endpoint local optimum is selected and the x-axis is the distance removed from this local optimum. Again the thickness is the weight of the edge, the amount of times it occurs. The radius of a node is the amount of times that node is visited. Edges with weight 1 are removed (otherwise it is a big blue blob).

# 12 Genetic Algorithms & Crossover Operators

In this section I will investigate crossovers. The crossovers I will use are CX, MPX, OX, PMX and EAX (see Section 3.2 for detailed explanation). GPX will not be used (see next section). To investigate the performance of these crossovers I will look at crossover correlation and their performance as the crossover in a genetic local search algorithm. In order to answer the following research question:

**How well do different crossover operators perform on TTP?**

Besides doing crossover on the tour I will also look at crossover on the packing plan. Since this is a bitstring we can use a normal crossover operator, for example two-point crossover. The question remains if doing a crossover on the tour and also on the bitstring is even necessary since it might disrupt the solution too much.

## 12.1 A Note on GPX for TTP

I have implemented GPX(2) with the three enhancements of Sanches, Whitley, and Tinós [58]. Unfortunately I found that the crossover was not always applicable. I ran some experiments testing GPX with local optima produced by 2-Opt with TSP and local optima produced by my local search with TTP. In total I have produced 1000 local optima for TSP and TTP and tried to apply GPX.

Results are displayed in Figure 34. GPX is far less suitable for TTP local optima than for TSP local optima. The amount of times GPX is applicable is much lower for TTP than for TSP. The total amount of viable partitions is much lower for TTP than for TSP. This can be due to the fact that there is a high variety in tours found by the TTP local search, for example much longer tours are found. I suspect that GPX might work even better for local optima produced by LKH than produced by 2-Opt.

Eventually with a higher problem size the probability that GPX is applicable will increase. I however only focus on the smaller instances and therefore I will not use GPX as crossover for the following crossover experiments.

## 12.2 Genetic Algorithms

In order to test which crossover performs best I have performed multiple experiments by running different genetic algorithms on the 9 instances previously studied. Besides testing the different crossovers I will also experiment with using two-point crossover on the packing plan and resetting the packing plan all together with a greedy packing heuristic of Faulkner et al. [16].

For the genetic algorithm I use a generational GA where selection is done by replacement. The offspring competes against the previous generation with (N+N)-selection. Each offspring is made by first applying the chosen tour crossover and using the chosen packing plan strategy. See Algorithm 9 for

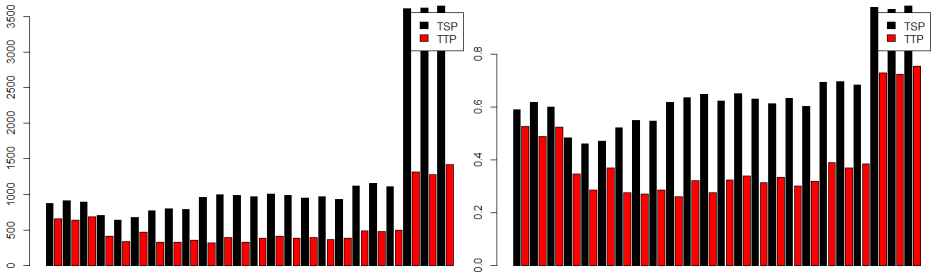


Figure 34: Two plots about the applicability of GPX for TSP (black) and TTP (red). Left plot shows the amount of partitions in 1000 tries and the right plot shows the ratio GPX was applicable in 1000 tries. The graphs shows instances of type $BSC05$, $USW05$ and $U05$ of the TSP instances: $eil51$, $berlin52$, $pr76$, $kroA100$, $bier127$, $ch130$, $rat195$ and $a280$. Ordered by problem size from left to right.

the full procedure. The following experiment will focus mostly on the amount of times the global optimum is found by using 2500 local searches in total.

### 12.2.1 Results

Results are in Table 16. First thing to note is that compared to ILS the global optimum is found far more often. But this is to be expected. As we just learned from the previous chapter, some instances have a multi-funnel structure. Since GA's are population based they can search at multiple locations in the search space and therefore wouldn't necessarily get stuck in one of those funnels.

Best results are obtained by EAX with doing two point crossover on the packing plan. The good performance of EAX can be due to the fact that EAX is by far the most sophisticate crossover of the lot. While others only manipulate permutations, EAX uses the structure of the graph of the tours to create offspring.

For the packing plan strategies two point crossover seems to be the best. This is interesting since combining a crossover on the tour and packing plan can also be viable . The offspring could inherent the status of the packing plan from the parent for which city was copied [14, 41]. This can be trivially be implemented for CX but nor for EAX.

Only doing crossover on the tour results in one of the worst results. But, for some reason the combination of no crossover on the packing plan and partial matched crossover work reasonable well. This could imply PMX is a good perturbation on the tour. While the others work better in combination with a packing plan strategy and try to combine solutions to find a better solution.

Of the classical operators CX performs best while MPX and OX have the worst performance. This could come from the fact that cycle crossover prioritizes absolute position which is important in the TTP and MPX and OX do not. Funny enough, the crossovers MPX and OX, which are quite similar, are used in the literature and CX is not (Figure 2).

**Algorithm 9** Memetic Algorithm

```
1: function GA
2:     Pop ← an initial population
3:     while k ≤ MaxGenerations do
4:         for i ← 1 to n do
5:             parent_1 ← Select random parent
6:             parent_2 ← Select random parent
7:
8:             child_i ← Create new child
9:             child_{i_Π} ← crossover(parent_{1_Π}, parent_{2_Π})
10:            child_{i_Y} ← 2PX(parent_{1_Y}, parent_{2_Y})      ▷ or Packing Heuristic
11:                                                  ▷ or inherent packing plan from parent_1
12:            child_i ← LocalSearch(child_i)
13:
14:            Pop ← Pop ∪ {child_i}
15:        end for
16:        Pop ← Sort population and keep n solutions
17:        k ← k + 1
18:    end while
19: end function
```

## 12.3 Crossover Correlation

For every tour crossover and packing plan strategy I have also calculated the crossovers correlation. I have produced 10.000 pairs of local optima and for each pair applied the crossover and packing plan strategy. On the offspring I have performed local search and calculated how much the fitness value of these children correlate with both parents.

Results are in Figure 35. For some crossovers and packing plan strategies there is some correlation between parent and offspring. This holds for EAX and especially for MPX where the correlation is high for every packing plan strategy. Other crossovers have little, no or a small negative correlation (CX and OX for an example). It is remarkable that MPX has one of the highest correlations and OX one of the smallest. Remarkable since the procedures of both are quite similar. This difference may be due to the fact that the TTP requires that the tour starts at city 0. In order to facilitate this constraint the tour must be repaired after some crossovers. This in turn can have a huge impact on the absolute position and might make the difference in performance of MPX and OX.

The predictive power of the crossover correlation in this context maybe non existent. According to the crossover correlation MPX should perform best but on the contrary, it performs worst. There does however seems to be a small negative correlation between the amount of times a non optimal solution is found and the crossover correlation (Figure 36). Crossover correlation is maybe the wrong measure to use for memetic algorithms for two reasons:

Table 16: Table showing per combination of TTP, crossover and packing plan strategies how many times the global optimum was **not** found.

| Crossover | | BSC | | | USW | | | U | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $Y$ | $\Pi$ | 01 | 05 | 10 | 01 | 05 | 10 | 01 | 05 | 10 | $\Sigma$ |
| - | | | | | | | | | | | |
| | EAX | 0 | 33 | 53 | 1 | 3 | 5 | 37 | 0 | 35 | 167 |
| | CX | 64 | 13 | 44 | 0 | 0 | 0 | 21 | 0 | 0 | 142 |
| | MPX | 71 | 11 | 62 | 0 | 0 | 0 | 25 | 0 | 1 | 170 |
| | OX | 71 | 32 | 45 | 0 | 1 | 0 | 48 | 0 | 2 | 199 |
| | PMX | 0 | 4 | 33 | 0 | 0 | 2 | 13 | 0 | 18 | 70 |
| 2PX | | | | | | | | | | | |
| | EAX | 4 | 2 | 9 | 0 | 0 | 1 | 18 | 0 | 1 | 35 |
| | CX | 11 | 2 | 14 | 0 | 0 | 0 | 22 | 0 | 1 | 50 |
| | MPX | 5 | 55 | 31 | 4 | 24 | 2 | 39 | 7 | 8 | 175 |
| | OX | 1 | 29 | 43 | 0 | 1 | 8 | 44 | 0 | 35 | 161 |
| | PMX | 1 | 18 | 44 | 1 | 0 | 8 | 43 | 0 | 41 | 156 |
| | | | | | | | | | | | |
| Greedy | EAX | 1 | 3 | 40 | 0 | 0 | 0 | 10 | 0 | 13 | 67 |
| | CX | 1 | 16 | 42 | 0 | 0 | 1 | 13 | 0 | 7 | 80 |
| | MPX | 0 | 11 | 59 | 0 | 0 | 1 | 26 | 0 | 15 | 112 |
| | OX | 0 | 12 | 56 | 0 | 0 | 3 | 25 | 0 | 16 | 112 |
| | PMX | 0 | 17 | 51 | 0 | 0 | 1 | 37 | 0 | 25 | 131 |

- Local search is done on the offspring which mutates the solution
- Highly crossover correlation can also be produced by offspring that is too similar to the parent(s)

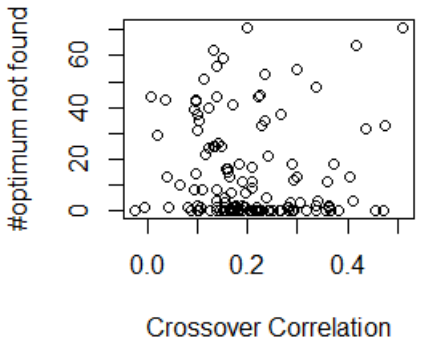


Figure 36: Plot showing the relation between crossover correlation and the number of times the optimum is not found.

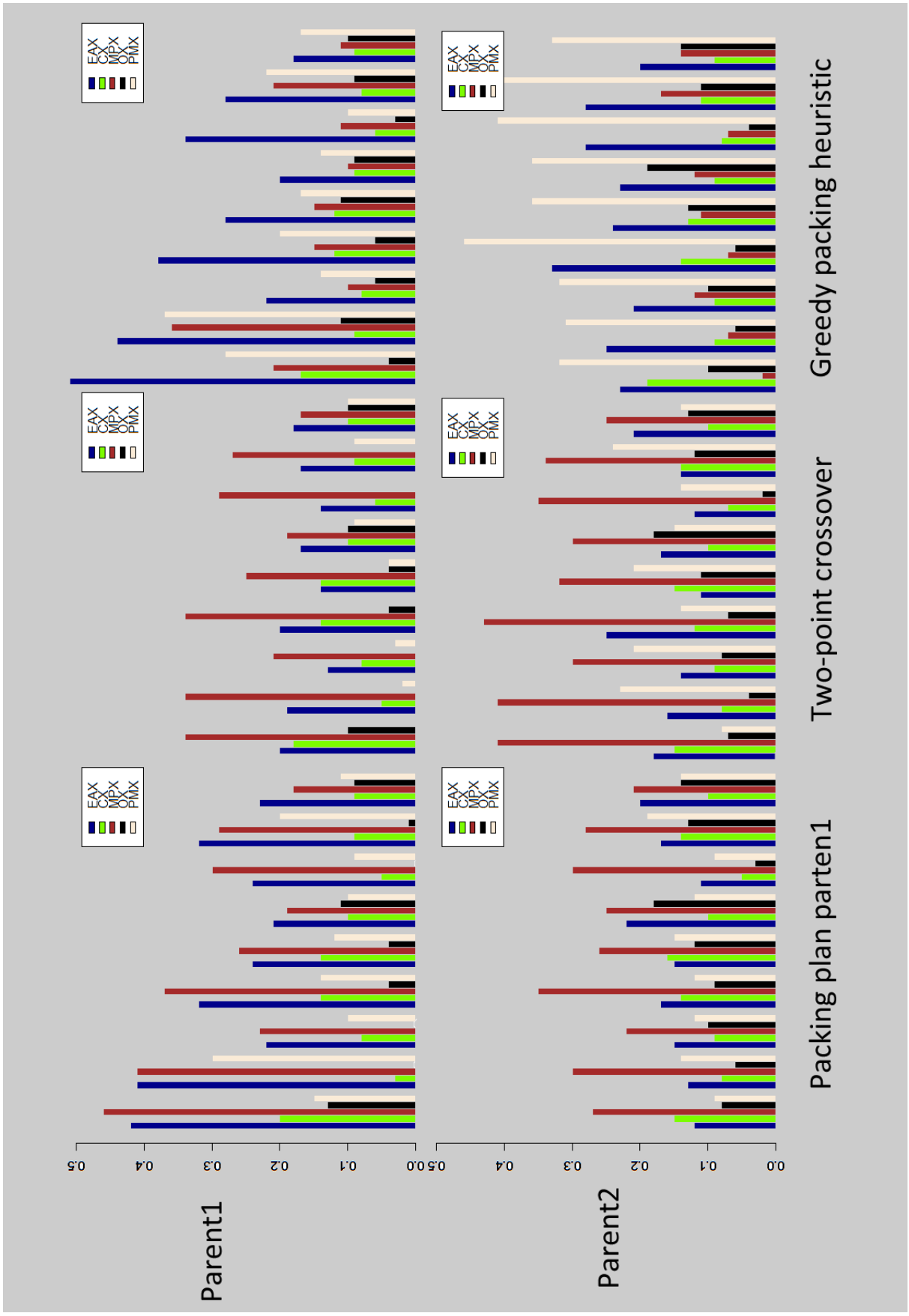


Figure 35: Crossover correlation for all crossover and packing strategies

## 12.4 Steady State Genetic Algorithm

When doing the previous experiments I found that a generational GA where selection is done by replacement converged rather quick. A fast convergent can have the same problems as an ILS since the search becomes narrow quickly. Thereby effectively removing the benefits of a population based metaheuristic. In order to account for this I ran additional experiments but now I only added offspring into the population if they were unique. For convenience I used a steady state algorithm instead of a generational GA in order to accommodate this constraint.

At each "generation" the steady state algorithms produce only one offspring and if it is better it replaces the worst member of the current population. For all steady state algorithms I used two point crossover on the packing plan. The pseudocode is in Algorithm 10.

**Algorithm 10** Steady State

```
1:  function GA
2:      Pop ← an initial population
3:      while k ≤ MaxGenerations do
4:          parent_1 ← Select random parent
5:          parent_2 ← Select random parent
6:
7:          child_i ← Create new child
8:          child_iΠ ← crossover(parent_1Π, parent_2Π)
9:          child_iΥ ← 2PX(parent_1Υ, parent_2Υ)
10:
11:         if Population does not contain child_i then
12:             Pop ← Replace member with lowest fitness with child_i
13:         end if
14:         k ← k + 1
15:     end while
16: end function
```

### 12.4.1 Results

Results are in Table 17. These results of the steady state algorithm are even better. In only 5 out of 900 runs the steady state genetic algorithm with EAX could not find the optimal solution. Some similar results can be seen for the other crossovers. MPX performs the worst by far. PMX seems to perform much better absolute and relative to other crossovers compared to its generational variant.

## 12.5 Crossover Experiment

As previously mentioned, the strength of a genetic algorithm can lie in the fact that it is population based. Combined with crossovers that alter the solutions it

Table 17: Table showing per combination of TTP instance and crossover how many times the global optimum was **not** found.

| | BSC | | | USW | | | U | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Crossover | 01 | 05 | 10 | 01 | 05 | 10 | 01 | 05 | 10 | Σ |
| EAX | 3 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |
| CX | 3 | 2 | 18 | 0 | 0 | 0 | 2 | 0 | 2 | 27 |
| MPX | 35 | 1 | 17 | 0 | 0 | 0 | 12 | 0 | 0 | 65 |
| OX | 9 | 1 | 11 | 0 | 1 | 0 | 0 | 0 | 0 | 22 |
| PMX | 6 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 15 |

is inherently some sort of multistart iterated local search. In order to investigate if the crossover not only perturb the solution but also inherent some of the good building blocks, I ran additional experiments where I replaced the crossover with a perturbation.

Again the perturbation will be $k$ random applications of the 2-opt swap and $k$ random bit flips. I will test with the best performing perturbation size of 5 and a smaller one of 3 since the exploitative power of a high perturbation size could also be account for by the population (and therefore a smaller perturbation size might perform better). I will also vary with the population size but keep the total local searches at a constant of 2500.

Results are in Table 18. I had suspected that this would perform much better than using the crossovers but it didn't. Apparently the used crossovers in the genetic algorithms do much more than just perturbing the solutions. For the generational GA the ILS-mutations are outperformed by EAX and CX and for the steady state algorithm each crossover operator outperforms the variant with perturbation.

It also seems to be the case that for some procedures a smaller perturbation size is better. For example the steady state algorithm with a perturbation size of 3 finds the global optimum more often than with a perturbation size of 5.

Table 18: Table showing how many times the global optimum was **not** found for different TTP instances and variants of genetic local searches with only mutations.

| | | | BSC | | | USW | | | U | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | peturb | gen x popsize | 01 | 05 | 10 | 01 | 05 | 10 | 01 | 05 | 10 | Σ |
| (N+N) | 3 | 50 x 50 | 0 | 29 | 52 | 0 | 0 | 0 | 12 | 0 | 3 | 96 |
| | 3 | 25 x 100 | 0 | 18 | 65 | 0 | 0 | 0 | 6 | 0 | 2 | 91 |
| | 3 | 100 x 25 | 0 | 40 | 42 | 0 | 1 | 0 | 25 | 0 | 3 | 111 |
| | 5 | 50 x 50 | 0 | 25 | 68 | 0 | 0 | 0 | 9 | 0 | 1 | 103 |
| | 5 | 25 x 100 | 0 | 34 | 65 | 0 | 0 | 0 | 11 | 0 | 1 | 111 |
| | 5 | 100 x 25 | 0 | 35 | 68 | 0 | 0 | 0 | 9 | 0 | 1 | 113 |
| SS | 3 | | 54 | 0 | 18 | 0 | 0 | 0 | 54 | 0 | 0 | 126 |
| | 5 | | 93 | 8 | 36 | 42 | 0 | 0 | 85 | 0 | 0 | 264 |

## 12.6 Conclusions

**How well do different crossover operators perform on TTP?**

Edge assembly crossover seems to perform the best in combination with two point crossover. Of the classical operators cycle crossover for the generational GA seems to perform best. For the steady state GA's all classical operators beside MPX perform the same and all find the global optimum with a high reliability. Partial mapped crossover with no packing plan strategy might be a viable option as a perturbation operator.

The genetic algorithms with their respective crossovers seem to really add something in trying to solve TTP. Especially by looking at the last experiments it is safe to conclude GA outperforms ILS. This is easy to see if you compare Table 17 and Table 12. GA finds the global optima with much more consistency. Also note that GA uses 2500 local searches and ILS at least 2500 local searches, in other words they use similar resources. But not only that, the crossover seems to navigate the search space better than a simple perturbation.

Moreover the use of a genetic algorithm can also be justified by the fact that the additional computational overhead is relatively non-existent since the local search of TTP is already computational heavy.

Crossover correlation was not as insightful as initially presumed. The crossover with the highest crossover correlation didn't performed the best. The best performing crossover, EAX, however did have one of the highest crossover correlations.

For these small instances it seems better to only add unique solutions to the population in order to avoid a fast convergence. Therefore the use of a steady state genetic algorithm with this criteria seems to be justified.

# 13 Comparison and Quality of Solutions

The last section of my thesis I will use the best found algorithm of the previous chapter on a variety of instances and compare them to the best found in the literature. I will also review the quality of the produced solutions. The best found solutions are from a case study of Wagner et al. [64]. Wagner et al. ran 21 algorithms on all instances for 10 min. The 21 algorithm are: SH [56], DH [8], (1+1)-EA [56], RLS [56], S1-S5 [16], C1-C6 [16], MALTS [36], CS2SA [14][22] and four different configurations of MMAS [63]. To my best knowledge there is no other published database of best found solutions[23].

The best algorithm I found was the steady state algorithm with EAX and 2PX with k-quadrant nearest neighborhood reduction.

I will run the algorithm on instances of:

- $eil51$
- $berlin52$
- $pr76$
- $kroA100$
- $bier127$
- $ch130$
- $rat195$
- $a280$

For instance of $eil51$, $berlin52$ and $pr76$ I will look at all item factors and only at item factor 1 for the others.

I will run the algorithm 10 times and every time use a limit of 2500 local searches. This is different from the 10 minute limit that Wagner et al. [64] apply. To give an impression how this compares, on my computer[24] the time it takes to perform 2500 local searches on the smallest instance ($eil51$) is around 9 seconds and less than 23 minutes for the largest ($a280$). For all instances except $a280$ the running time was below 10 minutes (see Appendix A.5 Table 27 for all the times).

In order to evaluate the quality of the solution I also looked at how many times a solution could still be improved with dynamic programming. This however is not part of the produced solution when comparing it to the best found.

## 13.1 Results

A long table of results is in the Appendix A.5 Table 28. Table 19 and Table 20 give a good impression of the results from the instances with an item factor of 1. In all but two instances equal or better than the best found solution is found. In almost all instances a new best solution is found.

Table 21 shows how many times the packing plan could be improved by dynamic programming. These results are similar to the findings of section 8. It seems that BitFlip and Exchange are highly capable of finding the best packing for a given tour when the knapsack type is $NOT$ correlated. In only 1 out of 1600 runs the produced solution could be improved with dynamic

[22] I have excluded fitness scores found by CS2SA since there are produced incorrectly (see section 4.7)

[23] There does however sometimes exists some papers [14, 62, 41, 3] who claim to have even better fitness scores than those of [64] or those presented in this thesis but as explained in section 4.7 they make rounding errors and are therefore incorrect.

[24] C#, i7-2600K 3.40 GHz, running on Windows 10

programming for the uncorrelated types. However for the larger instances the solutions of type bounded strongly correlated could be improved with dynamic programming plenty of times (82 out of 800 runs). For only 3 instances[25] the best run out of 10 could be improved with dynamic programming. These are interesting results, it seems that the instances of type bounded strongly correlated are harder to solve. This isn't analytically evident, the strong correlation between profit and weight partly disappears by the added hidden cost of an item's weight.

I'm fairly confident that for the smallest instance ($eil51$) the found solutions are optimal. I base this on the fact that these solutions are found with high reliability and by many different approaches: the "global" optima can be found with a multi-start local search algorithm with a small probability, with iterated local search this probability already becomes a lot higher and with a genetic local search in almost all the times. The fact that some of these best found solutions correspond to the best found in the literature even further confirms this in my opinion.

I cannot however make these claims for instances with a higher number of cities. On the contrary, I believe that for instances with a high number of cities a better solution can still be found. Table 20 shows that for these instances the best solution is only found once or never.

Table 22 and 23 show the results of instances with an item factor of more than one. While this thesis didn't focus on instances with more than one item per city, there is no reason why these results aren't applicable on instances with a larger item factor. Again similar results are found, for all but one instance the best found solution has a fitness value equal or greater than the current best in the literature. While the number of decisions variable increases by a lot the algorithm still produces good solutions. However, it seems like the algorithm has a hard time solving instances with a high item factor and where profit and weight are strongly correlated.

Of almost all the instances the average of the 10 runs is also better than the best in the literature. Even the worst of the 10 is most of the time better (of detailed results see Appendix A.5 Table 28). But to be fair, most algorithms in the case study of Wagner et al. [64] aren't sophisticated and/or are specifically designed for larger instances. In order to solve larger instances a lot of corners have to be cut. MATLS for example use fitness approximation to increase running time but this obviously decreases quality of found solutions. Therefore the fact that my genetic algorithm found better solutions is not incredibly impressive.

[25] BSC01 BSC04 of $bier127$ and BSC02 of $rat195$

Table 19: Table showing for every studied instance when the best solution of the steady state algorithm is better (green +), the same (yellow 0) or worse (red -) than the best found solution in the literature.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| eil51 | BSC | 0 | 0 | 0 | + | + | + | + | + | + | + |
| | U | + | + | 0 | 0 | 0 | 0 | + | + | + | + |
| | USW | + | + | + | + | + | 0 | + | + | + | 0 |
| berlin52 | BSC | + | + | + | + | + | + | + | + | + | + |
| | U | + | + | + | + | + | 0 | 0 | 0 | 0 | + |
| | USW | + | + | + | + | + | + | + | 0 | 0 | + |
| pr76 | BSC | + | 0 | + | + | + | + | + | + | + | + |
| | U | + | + | + | + | + | + | + | + | + | + |
| | USW | + | + | + | + | + | + | + | + | + | + |
| kroA100 | BSC | + | + | + | + | + | + | + | + | + | + |
| | U | + | + | + | + | + | + | + | + | + | + |
| | USW | + | + | + | + | + | + | + | + | + | + |
| bier127 | BSC | + | + | + | + | + | + | + | + | + | + |
| | U | + | + | + | + | + | + | + | + | + | + |
| | USW | + | + | + | + | + | + | + | + | + | + |
| ch130 | BSC | + | - | - | + | + | + | + | + | + | + |
| | U | + | + | + | + | + | + | + | + | + | + |
| | USW | + | + | + | + | + | + | + | + | + | + |
| rat195 | BSC | + | + | + | + | + | + | + | + | + | + |
| | U | + | + | + | + | + | + | + | + | + | + |
| | USW | + | + | + | + | + | + | + | + | + | + |
| a280 | BSC | + | + | + | + | + | + | + | + | + | + |
| | U | + | + | + | + | + | + | + | + | + | + |
| | USW | + | + | + | + | + | + | + | + | + | + |

Table 20: The amount of times out of 10 the algorithm corresponded to the best found solution

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| eil51 | BSC | 10 | 10 | 10 | 9 | 10 | 8 | 10 | 10 | 5 | 9 |
| | USW | 10 | 9 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| | U | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| berlin52 | BSC | 10 | 10 | 5 | 9 | 1 | 10 | 10 | 10 | 10 | 10 |
| | USW | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| | U | 9 | 10 | 10 | 9 | 10 | 10 | 10 | 10 | 10 | 10 |
| pr76 | BSC | 9 | 10 | 8 | 4 | 3 | 9 | 5 | 7 | 4 | 8 |
| | USW | 10 | 10 | 10 | 9 | 10 | 10 | 10 | 10 | 10 | 10 |
| | U | 10 | 10 | 10 | 8 | 8 | 7 | 10 | 10 | 10 | 10 |
| kroA100 | BSC | 10 | 7 | 3 | 3 | 1 | 6 | 3 | 4 | 4 | 5 |
| | USW | 9 | 10 | 10 | 9 | 9 | 8 | 6 | 7 | 8 | 7 |
| | U | 9 | 10 | 10 | 10 | 10 | 10 | 9 | 7 | 9 | 6 |
| bier127 | BSC | 1 | 1 | 2 | 2 | 1 | 2 | 5 | 1 | 6 | 5 |
| | USW | 7 | 2 | 7 | 1 | 1 | 3 | 2 | 1 | 8 | 6 |
| | U | 1 | 1 | 2 | 1 | 4 | 5 | 5 | 9 | 8 | 8 |
| ch130 | BSC | 1 | 0 | 0 | 1 | 6 | 2 | 3 | 4 | 5 | 8 |
| | USW | 4 | 6 | 1 | 5 | 2 | 4 | 6 | 3 | 1 | 5 |
| | U | 3 | 2 | 1 | 1 | 2 | 1 | 2 | 5 | 5 | 3 |
| rat195 | BSC | 1 | 1 | 2 | 1 | 1 | 1 | 2 | 1 | 2 | 1 |
| | USW | 1 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 |
| | U | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| a280 | BSC | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | USW | 1 | 1 | 3 | 1 | 1 | 1 | 1 | 5 | 4 | 1 |
| | U | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 |

Table 21: The amount of times out of 10 the produced solution could be improved by dynamic programming.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| eil51 | BSC | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | U | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | USW | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| berlin52 | BSC | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| | U | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | USW | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| pr76 | BSC | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| | U | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | USW | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| kroA100 | BSC | 0 | 1 | 2 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| | U | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | USW | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| bier127 | BSC | 4 | 3 | 3 | 9 | 2 | 2 | 2 | 0 | 1 | 0 |
| | U | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | USW | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ch130 | BSC | 0 | 5 | 3 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| | U | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | USW | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| rat195 | BSC | 3 | 1 | 1 | 0 | 2 | 3 | 2 | 1 | 0 | 0 |
| | U | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | USW | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| a280 | BSC | 8 | 3 | 5 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| | U | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | USW | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 22: Table showing for every studied instance if the best solution of the steady state algorithm is better (green +), the same (yellow 0) or worse (red -) than the best found solution in the literature.

| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| eil51 | 3 | BSC | + | + | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | U | 0 | + | 0 | + | 0 | + | + | 0 | 0 | 0 |
| | | USW | + | 0 | 0 | + | + | 0 | 0 | 0 | + | + |
| | 5 | BSC | + | + | + | + | 0 | 0 | 0 | 0 | 0 | 0 |
| | | U | + | + | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | USW | + | 0 | 0 | 0 | 0 | 0 | + | 0 | 0 | + |
| | 10 | BSC | 0 | + | + | 0 | + | 0 | + | 0 | + | + |
| | | U | + | + | + | + | 0 | + | 0 | 0 | 0 | 0 |
| | | USW | + | 0 | 0 | + | + | 0 | 0 | 0 | 0 | 0 |
| berlin52 | 3 | BSC | + | + | + | + | + | 0 | + | + | + | + |
| | | U | 0 | + | + | 0 | 0 | + | 0 | 0 | + | + |
| | | USW | + | + | + | + | + | + | + | + | + | + |
| | 5 | BSC | + | + | + | + | + | - | + | + | + | + |
| | | U | + | + | + | + | + | + | + | + | + | + |
| | | USW | + | + | + | + | + | + | + | + | + | + |
| | 10 | BSC | + | + | + | + | + | + | + | + | + | + |
| | | U | + | + | + | + | + | + | + | + | + | + |
| | | USW | + | 0 | + | + | + | + | + | + | + | + |
| pr76 | 3 | BSC | + | + | + | + | + | + | + | + | + | + |
| | | U | + | + | + | + | + | + | + | + | + | + |
| | | USW | + | + | + | + | + | + | + | + | + | + |
| | 5 | BSC | + | + | + | + | + | + | + | + | + | + |
| | | U | + | + | + | + | + | + | + | + | + | + |
| | | USW | + | + | + | + | + | + | + | + | + | + |
| | 10 | BSC | + | + | + | + | + | + | + | + | + | + |
| | | U | + | + | + | + | + | + | + | + | + | + |
| | | USW | + | + | + | + | + | + | + | + | + | + |

Table 23: The amount of times out of 10 the algorithm corresponded to the best found solution

| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| eil51 | 3 | BSC | 10 | 10 | 4 | 9 | 10 | 10 | 10 | 10 | 10 | 10 |
| | | U | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| | | USW | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| | 5 | BSC | 6 | 10 | 10 | 10 | 8 | 9 | 5 | 10 | 10 | 10 |
| | | U | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| | | USW | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| | 10 | BSC | 6 | 6 | 1 | 7 | 10 | 10 | 9 | 10 | 5 | 10 |
| | | U | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| | | USW | 10 | 8 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| berlin52 | 3 | BSC | 5 | 2 | 6 | 4 | 2 | 1 | 10 | 10 | 10 | 10 |
| | | U | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| | | USW | 9 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| | 5 | BSC | 3 | 1 | 1 | 1 | 1 | 1 | 10 | 10 | 10 | 10 |
| | | U | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| | | USW | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| | 10 | BSC | 1 | 1 | 1 | 1 | 1 | 3 | 10 | 10 | 10 | 10 |
| | | U | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| | | USW | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| pr76 | 3 | BSC | 1 | 3 | 1 | 4 | 8 | 10 | 10 | 10 | 10 | 10 |
| | | U | 10 | 8 | 10 | 9 | 7 | 9 | 10 | 7 | 9 | 5 |
| | | USW | 3 | 10 | 10 | 8 | 9 | 9 | 10 | 10 | 9 | 9 |
| | 5 | BSC | 1 | 2 | 7 | 5 | 1 | 6 | 6 | 10 | 10 | 10 |
| | | U | 9 | 2 | 7 | 10 | 6 | 10 | 10 | 10 | 10 | 10 |
| | | USW | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| | 10 | BSC | 1 | 4 | 4 | 10 | 9 | 8 | 10 | 10 | 10 | 10 |
| | | U | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| | | USW | 1 | 6 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |

# 14 Conclusion

**How can the use and effect of various operators and strategies in the literature of the traveling thief problem be justified, explained and improved?**

The research question divided into sub-questions has been answered in their respected chapters throughout this thesis (see the summary in the next section). The main contributions of this master thesis are:

- Proposed a new local search procedure with two improvements to the computational complexity of the subprocedures INSERTION (improvement of $O(n)$) and EXCHANGE (improvement of $O(m)$) (section 7).
- Shown that a greedy packing heuristic can obtain optimal or near optimal solutions for the packing plan and why this is the case (section 8).
- Conducted a fitness landscape analysis which among other things shows that TTP instances contain a lot of local optima but their distance to the global optimum is correlated with its fitness (section 10).
- Generated a local optima network with respect to an iterated local search that shows TTP has a multi funnel structure (section 11).
- Conducted experiments that show that a steady state genetic algorithm with edge assembly crossover outperforms multi start local search, iterated local search and genetic algorithms with other tour crossovers (section 12).
- Found new best solutions to almost all studied instances of the benchmark suite [56] (section 13).

## 14.1 Summary

In this master thesis I have investigated various operators and strategies in the literature of the traveling thief problem in order to try to justify, explain and improve these operators and strategies. I have found the following results:

- Optimal or near optimal solutions for the packing plan can be obtained with a greedy packing strategy. With a high probability the optimal solution can be found with a packing heuristic (PACKING ITERATIVE [16] especially). The fitness function over multiple values of $\alpha$ is almost concave. The relatively low computational complexity of a greedy packing heuristic might imply the feasibility of it but it is uncertain how such a procedure should be combined with local search or in a meta heuristic. Similar high quality packing plans can be obtained with a combination of BITFLIP and EXCHANGE but with a higher time complexity.

- Neighborhood reduction strategies lowered the quality of the found local optima. But neighborhood reductions are necessary for the inclusion of 2-OPT. I have shown that 2-OPT is the bottleneck for the local search. There does however seems to be no neighborhood reduction that outperforms the others. Nearest neighbor, k-quadrant nearest neighbor and reduction by Delaunay triangulation do not perform significantly different. The quality of the local optima seems to correlate with the number of neighbors.

- INSERTION has a higher autocorrelation than 2-OPT for the TTP. The reverse is true for TSP. The fitness landscape of the TTP contains many unique local optima and the global optimum has a small (relative and absolute) region of attraction. The fitness distance correlation is high when the distance of tour and packing plans are combined. This is even higher when one of the subproblems is kept fixed.

- A local optima network with respect to an iterated local search for some TTP instances revealed a multi-funnel structure in this landscape.

- Iterated local search with $k$ random applications of the 2-opt swap and $k$ random bit flips can produce good results. ILS could find the global optimum in 50% to 100% of the cases.

- A steady state genetic algorithm with edge assembly crossover seems to perform really well. Only in 5 out of 900 runs the algorithm didn't find the global solution. The genetic algorithms with their respective crossovers seem to contribute in trying to solve TTP. The crossovers do not only perturb the solution they perform better and possibly inherit the good building blocks which produces good offspring.

- I have shown that iterated local search performs better than multistart local search which is coherent with the finding that the distance is highly correlated with the fitness. I have shown that genetic local search performs better than iterated local search which is coherent with the finding that TTP instances have a multi-funnel structure.

## 14.2 Discussion & Future work

In my master thesis I have been thinking extensively about the TTP. I have been watching the literature closely and experimented with different approaches. Now it is time to take a step back and look at TTP in a broader, maybe more speculative, perspective.

The traveling thief problem was created to study problems which consist of interdependent subproblems. The creators argue that for most real world problems their complexity is due to the fact that they consist of multiple interdependent subproblems [7]. They argue that this is the essential characteristic that makes a real world problem complex. By definition TTP is a problem consisting of two interdependent subproblems. But the question remains whether this interdependence is what makes problems hard.

The initial premise might be incorrect. Why should the fact that TTP consists of two separate interdependent subproblems make it a complex problem? Obviously the problem would be simpler if TTP consisted of two independent subproblems but is an TSP instance of equivalent input size any easier to solve?

The claim is that because of the interdependence of TTP the problem is indecomposable and should therefore by solved as a whole [37]. At the same time, contradictory, TTP is mostly solved by decomposing it into subproblems

[16, 36, 63]. A perfect decomposition of TTP is bound to fail, there is of course an interdependence. But is this unique to TTP? In every NP-hard problem there exists interdependence between the decisions variables. For every item added to a knapsack instance the optimal solution is potentially changed and this also holds for an added city to a TSP instance. But even for such problems decomposing can be beneficial. In my opinion the right way to effectively solve these problems is not to conclude that the problem is indecomposable. The right way would be to search for a decomposition that takes into account the interdependence of subproblems to exploit their independence.

It seems that one of the subproblems, the packing plan, is rather easy to solve[26] (section 8 & 13). If a benchmark problem with interdependent subproblems is of importance it would be probably valuable to make both subproblems hard to solve. Otherwise solving the whole problem can be done solely on the solution representation of the tour subproblem and by only considering the packing plan subproblem when evaluating a tour [63, 36]. What makes TTP hard? In my experience, the reason TTP is hard to *solve* with most metaheuristics is because it is impossible to do incremental fitness evaluation for the most trivial operators (section 7) which in turn makes finding good solutions for even the smallest instances a time consuming task (section 9). Incremental fitness evaluation is impossible because there is interdependence between decision variables and not necessarily because of the interdependence between the subproblems. Keeping one of the subproblems fixed still makes incremental fitness evaluation impossible for most operators.

It is evident that there exist real world problems which consist of multiple interdependent subproblems. TTP tries characterizing this and could indeed serve as a toy problem which could help to better understand these problems with interdependent subproblems. In order for the traveling thief problem to be justified as a subject of study of the essence of real world problems two things need to happen in future research:

- Confirm that consisting of interdependent subproblems is indeed the crucial factor of real world complexity
- Confirm that TTP (and the predominately studied benchmark instances) successfully captures this complex ity

This is not trivial, but it is crucial. Otherwise we would chase yet another benchmark problem while the supposed gap between research and practice in meta-heuristic methods, for which the problem was created, grows even further.

[26] at least for the instances that are predominantly studied

# A Appendix

## A.1

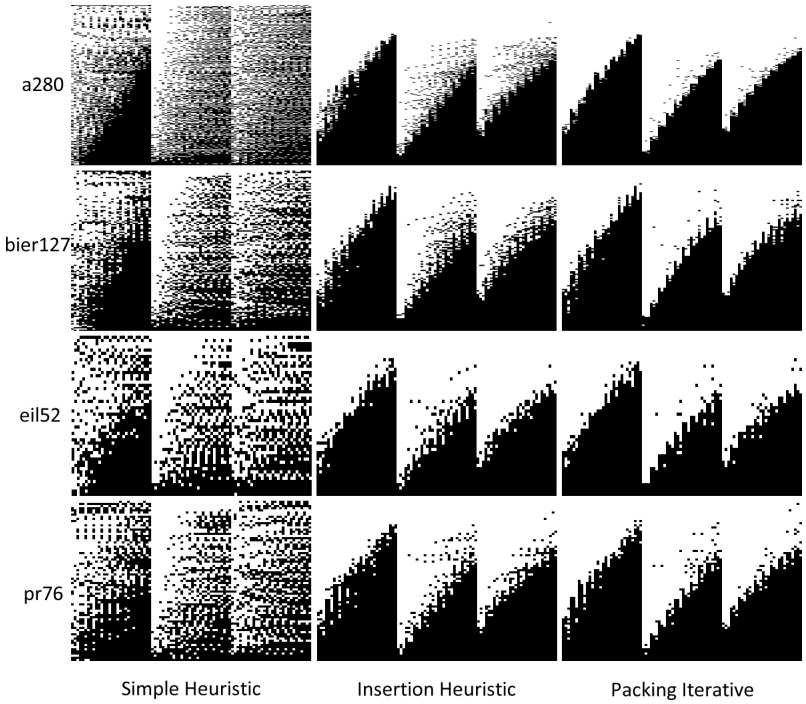


Figure 37: The performance of different packing heuristics visualized. See Figure 14 for an interpretation.

## A.2

Table 24: Table with the ordering of best average scores. Which also corresponds to the ordering of best median scores.

| | NN-4 | NN-8 | NN-16 | QN-4 | QN-8 | QN-16 | DT | DT2 | Normal |
|---|---|---|---|---|---|---|---|---|---|
| a280 | 8 | 5 | 3 | 7 | 4 | 1 | 6 | 2 | 0 |
| berlin52 | 8 | 5 | 1 | 7 | 4 | 2 | 6 | 3 | 0 |
| bier127 | 8 | 5 | 1 | 6 | 4 | 2 | 7 | 3 | 0 |
| ch130 | 8 | 5 | 3 | 7 | 4 | 2 | 6 | 1 | 0 |
| eil51 | 8 | 4 | 1 | 7 | 5 | 3 | 6 | 2 | 0 |
| kroA100 | 8 | 5 | 3 | 6 | 4 | 2 | 7 | 1 | 0 |
| pr76 | 8 | 5 | 1 | 6 | 4 | 3 | 7 | 2 | 0 |
| rat195 | 6 | 5 | 3 | 8 | 4 | 1 | 7 | 2 | 0 |

Table 25: Table with the average running time of every local search grouped by TSP-instances.

| | NN-4 | NN-8 | NN-16 | QN-4 | QN-8 | QN-16 | DT | DT2 | Normal |
|---|---|---|---|---|---|---|---|---|---|
| a280 | 2488 | 2771 | 3138 | 2579 | 2816 | 3190 | 2601 | 3182 | 17468 |
| berlin52 | 20 | 23 | 30 | 19 | 22 | 27 | 20 | 26 | 50 |
| bier127 | 262 | 292 | 355 | 259 | 288 | 339 | 260 | 334 | 1016 |
| ch130 | 266 | 295 | 340 | 261 | 292 | 337 | 265 | 338 | 1029 |
| eil51 | 17 | 19 | 26 | 16 | 19 | 23 | 17 | 23 | 46 |
| kroA100 | 123 | 138 | 164 | 126 | 140 | 162 | 126 | 161 | 419 |
| pr76 | 56 | 63 | 78 | 56 | 63 | 75 | 57 | 75 | 171 |
| rat195 | 841 | 933 | 1089 | 851 | 938 | 1090 | 873 | 1088 | 4901 |

## A.3

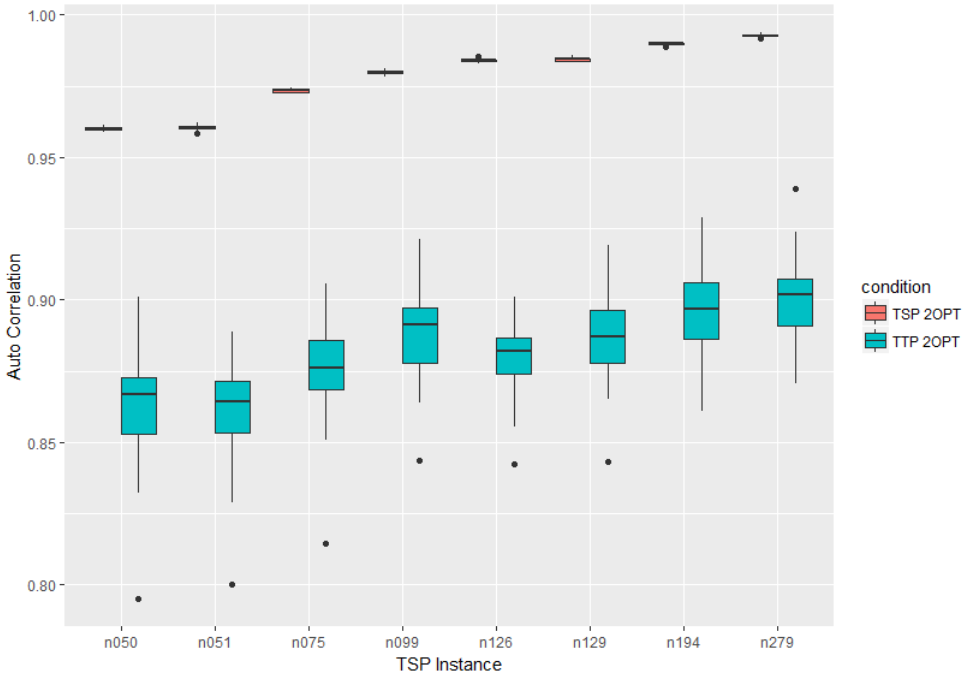


Figure 38: Auto correlation of 2-OPT for different TSP instances.

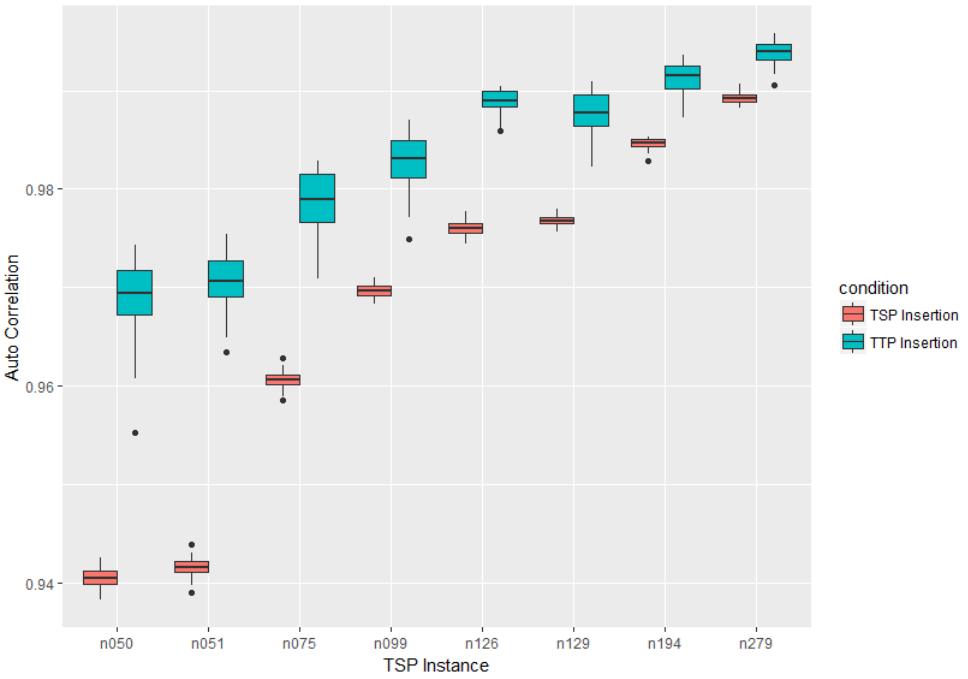


Figure 39: Auto correlation of INSERTION for different TSP instances.

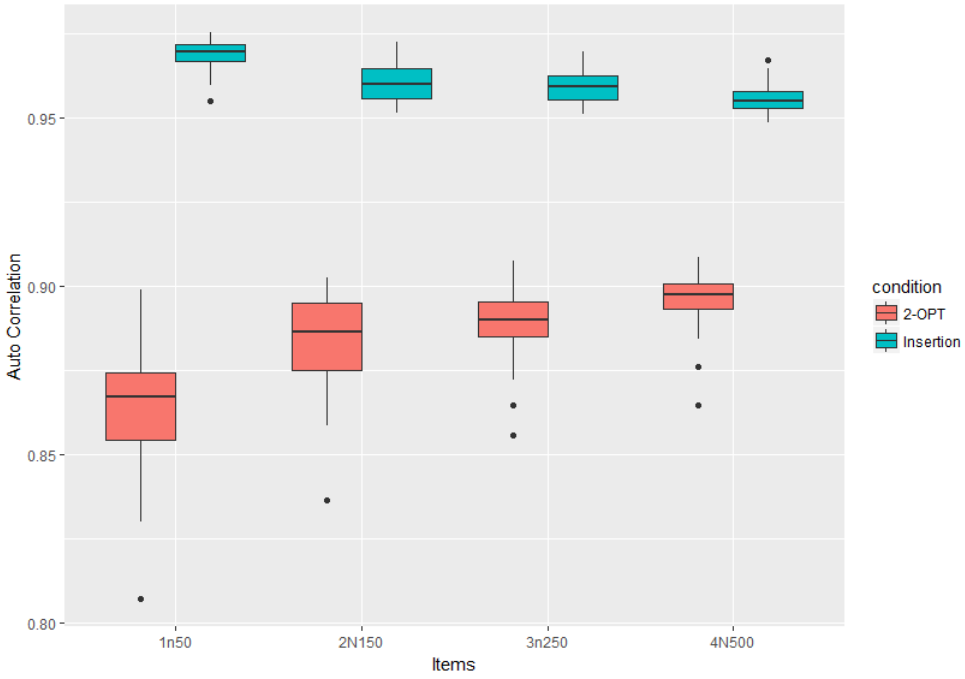


Figure 40: Auto correlation of tour operators for different item categories.

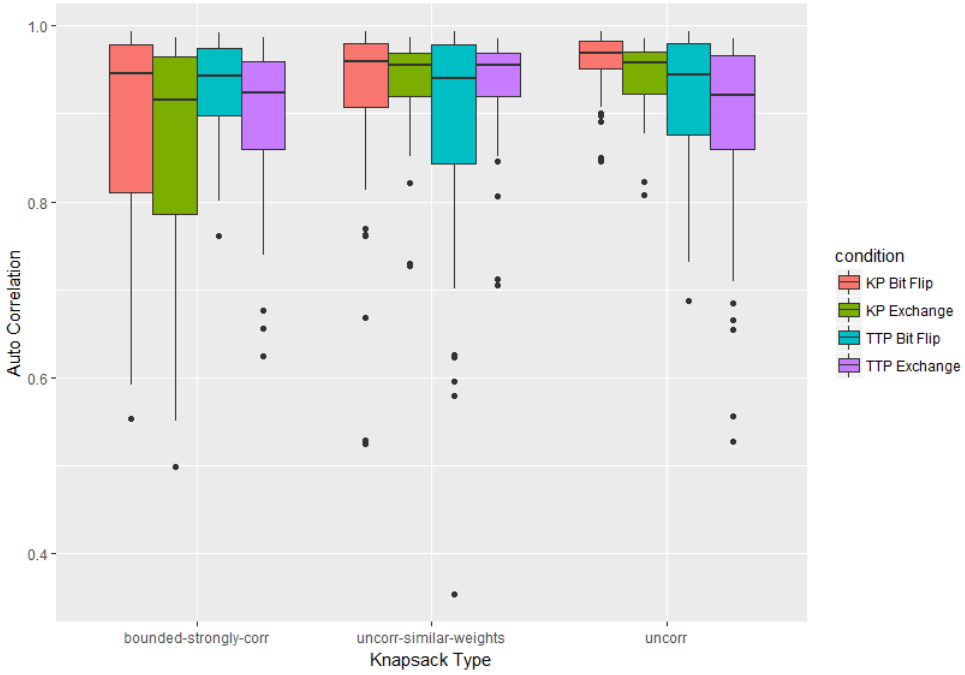


Figure 41: Boxplot of the autocorrelation of FLIP and EXCHANGE for all instances grouped by different knapsack types.

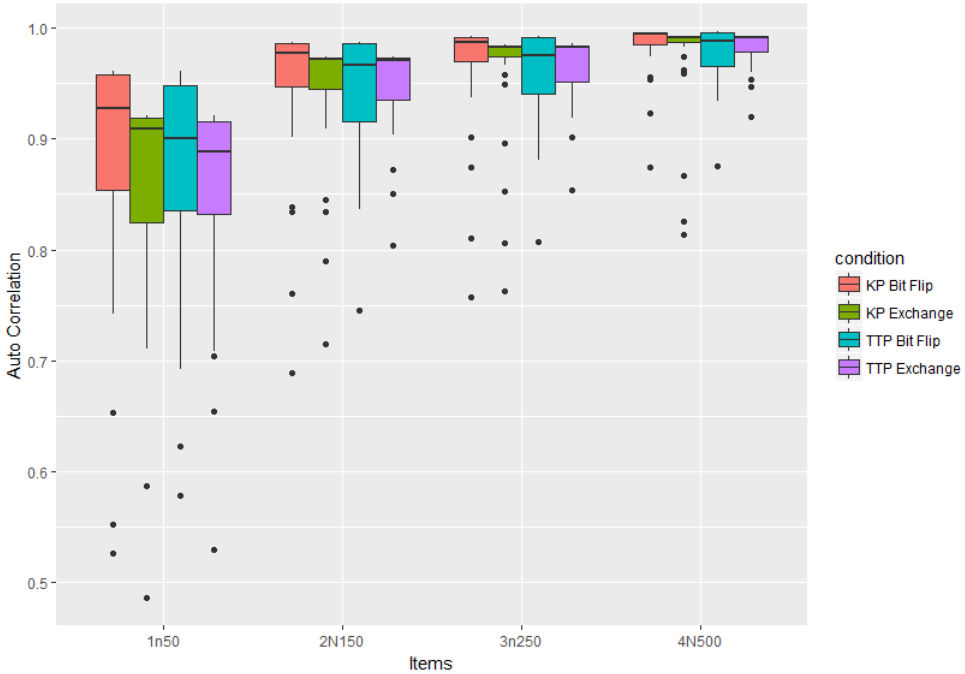


Figure 43: Auto correlation for different item categories on the KP and the TTP.

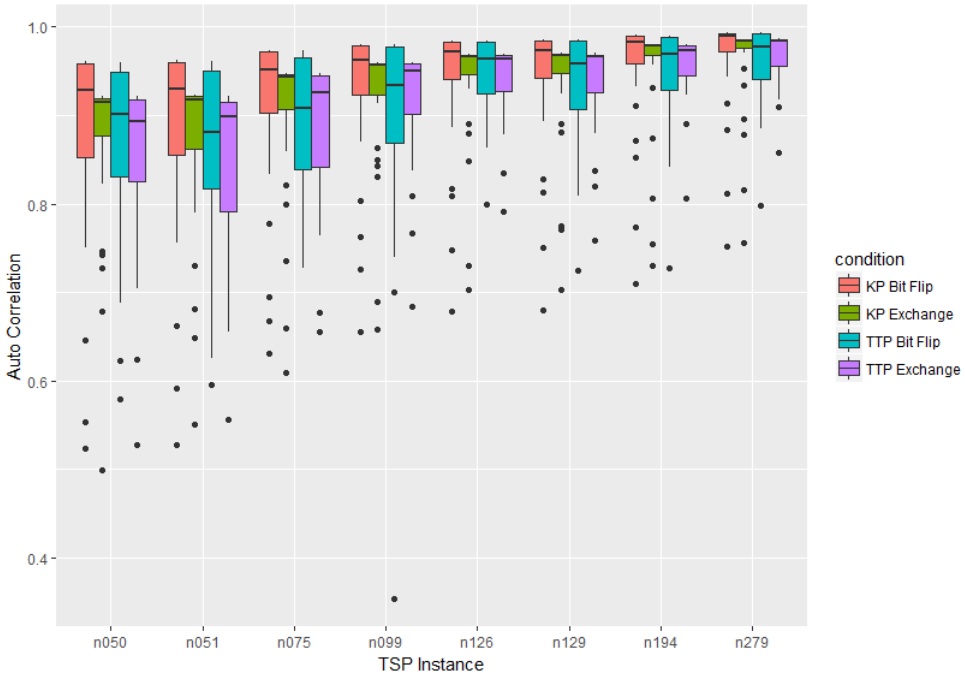


Figure 42: Auto correlation for different TSP instances on the KP and the TTP.

## A.4

Table 26: Median of the amount local optima produced before the final solution was found.

| | | 1 | 2 | 3 | 4 | 5 | 10 | 20 |
|---|---|---|---|---|---|---|---|---|
| BSC | | | | | | | | |
| | 01 | 671 | 422 | 274 | 168 | 144 | 306 | 563 |
| | 05 | 237 | 273 | 512 | 519 | 393 | 1047 | 1628 |
| | 10 | 417 | 790 | 781 | 1173 | 1265 | 1035 | 1357 |
| USW | | | | | | | | |
| | 01 | 176 | 85 | 88 | 99 | 106 | 158 | 294 |
| | 05 | 116 | 98 | 165 | 118 | 180 | 256 | 664 |
| | 10 | 178 | 120 | 135 | 142 | 145 | 319 | 527 |
| U | | | | | | | | |
| | 01 | 233 | 273 | 273 | 274 | 345 | 500 | 1315 |
| | 05 | 113 | 70 | 50 | 57 | 49 | 55 | 121 |
| | 10 | 163 | 131 | 346 | 357 | 338 | 444 | 552 |

## A.5

| | Benchmark | Best | Average | Worst |
|---|---|---|---|---|
| a280_n279_bounded-strongly-corr_01 | 18441 | 19499 | 19410 | 19214 |
| a280_n279_bounded-strongly-corr_02 | 32270 | 32768 | 32562 | 32237 |
| a280_n279_bounded-strongly-corr_03 | 40720 | 41589 | 41270 | 40831 |
| a280_n279_bounded-strongly-corr_04 | 49019 | 50184 | 49850 | 49189 |
| a280_n279_bounded-strongly-corr_05 | 55766 | 57990 | 57413 | 55616 |
| a280_n279_bounded-strongly-corr_06 | 49489 | 52186 | 51374 | 50657 |
| a280_n279_bounded-strongly-corr_07 | 56376 | 60127 | 59084 | 57598 |
| a280_n279_bounded-strongly-corr_08 | 53672 | 57989 | 57270 | 55968 |
| a280_n279_bounded-strongly-corr_09 | 58486 | 64513 | 63596 | 61447 |
| a280_n279_bounded-strongly-corr_10 | 58086 | 64402 | 63248 | 61829 |
| a280_n279_uncorr_01 | 19349 | 20491 | 20346 | 20220 |
| a280_n279_uncorr_02 | 28065 | 28775 | 28665 | 28382 |
| a280_n279_uncorr_03 | 34363 | 34727 | 34631 | 34498 |
| a280_n279_uncorr_04 | 34551 | 35504 | 35340 | 35125 |
| a280_n279_uncorr_05 | 32910 | 34203 | 34051 | 33657 |
| a280_n279_uncorr_06 | 37102 | 38666 | 38560 | 38391 |
| a280_n279_uncorr_07 | 36811 | 38414 | 38155 | 37044 |
| a280_n279_uncorr_08 | 39445 | 40943 | 40810 | 39833 |
| a280_n279_uncorr_09 | 40372 | 41523 | 41452 | 41336 |
| a280_n279_uncorr_10 | 42127 | 42931 | 42852 | 42735 |
| a280_n279_uncorr-similar-weights_01 | 9167 | 9998 | 9940 | 9873 |
| a280_n279_uncorr-similar-weights_02 | 12664 | 14237 | 14101 | 13952 |
| a280_n279_uncorr-similar-weights_03 | 17512 | 19095 | 18877 | 18692 |
| a280_n279_uncorr-similar-weights_04 | 19402 | 21163 | 20738 | 20204 |

| | | | | |
|---|---|---|---|---|
| a280_n279_uncorr-similar-weights_05 | 22277 | 24214 | 23846 | 23348 |
| a280_n279_uncorr-similar-weights_06 | 24500 | 26593 | 26296 | 25612 |
| a280_n279_uncorr-similar-weights_07 | 26007 | 27957 | 27625 | 26730 |
| a280_n279_uncorr-similar-weights_08 | 28517 | 30144 | 29810 | 28817 |
| a280_n279_uncorr-similar-weights_09 | 35573 | 36657 | 36502 | 36230 |
| a280_n279_uncorr-similar-weights_10 | 40820 | 41587 | 41434 | 41004 |
| berlin52_n51_bounded-strongly-corr_01 | 4411 | 4455 | 4455 | 4455 |
| berlin52_n51_bounded-strongly-corr_02 | 7072 | 7149 | 7149 | 7149 |
| berlin52_n51_bounded-strongly-corr_03 | 7831 | 7907 | 7905 | 7901 |
| berlin52_n51_bounded-strongly-corr_04 | 10810 | 10894 | 10894 | 10892 |
| berlin52_n51_bounded-strongly-corr_05 | 14173 | 14271 | 14244 | 14228 |
| berlin52_n51_bounded-strongly-corr_06 | 13785 | 13890 | 13890 | 13890 |
| berlin52_n51_bounded-strongly-corr_07 | 16604 | 16876 | 16876 | 16876 |
| berlin52_n51_bounded-strongly-corr_08 | 16964 | 17096 | 17096 | 17096 |
| berlin52_n51_bounded-strongly-corr_09 | 19558 | 19563 | 19563 | 19563 |
| berlin52_n51_bounded-strongly-corr_10 | 16399 | 16672 | 16672 | 16672 |
| berlin52_n51_uncorr_01 | 3089 | 3111 | 3111 | 3111 |
| berlin52_n51_uncorr_02 | 4435 | 4496 | 4496 | 4496 |
| berlin52_n51_uncorr_03 | 6249 | 6317 | 6317 | 6317 |
| berlin52_n51_uncorr_04 | 5926 | 6008 | 6008 | 6008 |
| berlin52_n51_uncorr_05 | 6450 | 6459 | 6459 | 6459 |
| berlin52_n51_uncorr_06 | 7704 | 7704 | 7704 | 7704 |
| berlin52_n51_uncorr_07 | 8647 | 8647 | 8647 | 8647 |
| berlin52_n51_uncorr_08 | 9351 | 9351 | 9351 | 9351 |
| berlin52_n51_uncorr_09 | 9847 | 9847 | 9847 | 9847 |
| berlin52_n51_uncorr_10 | 9643 | 9649 | 9649 | 9649 |
| berlin52_n51_uncorr-similar-weights_01 | 1625 | 1656 | 1656 | 1653 |
| berlin52_n51_uncorr-similar-weights_02 | 3781 | 3813 | 3813 | 3813 |
| berlin52_n51_uncorr-similar-weights_03 | 4998 | 5128 | 5128 | 5128 |
| berlin52_n51_uncorr-similar-weights_04 | 5184 | 5198 | 5198 | 5195 |
| berlin52_n51_uncorr-similar-weights_05 | 6746 | 6750 | 6750 | 6750 |
| berlin52_n51_uncorr-similar-weights_06 | 6661 | 6673 | 6673 | 6673 |
| berlin52_n51_uncorr-similar-weights_07 | 7456 | 7464 | 7464 | 7464 |
| berlin52_n51_uncorr-similar-weights_08 | 9110 | 9110 | 9110 | 9110 |
| berlin52_n51_uncorr-similar-weights_09 | 8854 | 8854 | 8854 | 8854 |
| berlin52_n51_uncorr-similar-weights_10 | 9059 | 9088 | 9088 | 9088 |
| bier127_n126_bounded-strongly-corr_01 | 5977 | 6125 | 6029 | 5945 |
| bier127_n126_bounded-strongly-corr_02 | 13338 | 13537 | 13467 | 13382 |
| bier127_n126_bounded-strongly-corr_03 | 16576 | 16860 | 16802 | 16753 |
| bier127_n126_bounded-strongly-corr_04 | 23558 | 23670 | 23639 | 23537 |
| bier127_n126_bounded-strongly-corr_05 | 25449 | 25650 | 25486 | 24993 |
| bier127_n126_bounded-strongly-corr_06 | 27985 | 28340 | 28113 | 27748 |
| bier127_n126_bounded-strongly-corr_07 | 34027 | 34890 | 34788 | 34298 |
| bier127_n126_bounded-strongly-corr_08 | 39712 | 40630 | 40496 | 40263 |
| bier127_n126_bounded-strongly-corr_09 | 39640 | 40982 | 40909 | 40596 |
| bier127_n126_bounded-strongly-corr_10 | 39355 | 40352 | 40308 | 40215 |

| | | | | |
|---|---|---|---|---|
| bier127_n126_uncorr_01 | 7747 | 8233 | 8231 | 8228 |
| bier127_n126_uncorr_02 | 9376 | 9841 | 9784 | 9638 |
| bier127_n126_uncorr_03 | 13855 | 14137 | 14096 | 13884 |
| bier127_n126_uncorr_04 | 15236 | 15382 | 15328 | 15247 |
| bier127_n126_uncorr_05 | 15876 | 15962 | 15906 | 15814 |
| bier127_n126_uncorr_06 | 17366 | 17667 | 17514 | 17396 |
| bier127_n126_uncorr_07 | 16964 | 17174 | 17073 | 16989 |
| bier127_n126_uncorr_08 | 17782 | 18046 | 18005 | 17987 |
| bier127_n126_uncorr_09 | 19898 | 20107 | 20105 | 20095 |
| bier127_n126_uncorr_10 | 20071 | 20260 | 20233 | 20180 |
| bier127_n126_uncorr-similar-weights_01 | 4679 | 4753 | 4740 | 4721 |
| bier127_n126_uncorr-similar-weights_02 | 5884 | 6217 | 6192 | 6167 |
| bier127_n126_uncorr-similar-weights_03 | 8636 | 9277 | 9206 | 9104 |
| bier127_n126_uncorr-similar-weights_04 | 10272 | 11020 | 10901 | 10741 |
| bier127_n126_uncorr-similar-weights_05 | 12552 | 13318 | 13309 | 13277 |
| bier127_n126_uncorr-similar-weights_06 | 14058 | 15016 | 14991 | 14962 |
| bier127_n126_uncorr-similar-weights_07 | 14960 | 16075 | 16025 | 15861 |
| bier127_n126_uncorr-similar-weights_08 | 17171 | 18049 | 18046 | 18026 |
| bier127_n126_uncorr-similar-weights_09 | 17975 | 19088 | 19043 | 18675 |
| bier127_n126_uncorr-similar-weights_10 | 18726 | 19705 | 19668 | 19368 |
| ch130_n129_bounded-strongly-corr_01 | 9516 | 9707 | 9659 | 9593 |
| ch130_n129_bounded-strongly-corr_02 | 16841 | 16834 | 16776 | 16591 |
| ch130_n129_bounded-strongly-corr_03 | 21619 | 21597 | 21546 | 21438 |
| ch130_n129_bounded-strongly-corr_04 | 25237 | 25533 | 25364 | 24926 |
| ch130_n129_bounded-strongly-corr_05 | 31888 | 32311 | 32257 | 32134 |
| ch130_n129_bounded-strongly-corr_06 | 34734 | 34988 | 34930 | 34785 |
| ch130_n129_bounded-strongly-corr_07 | 38207 | 38800 | 38546 | 38303 |
| ch130_n129_bounded-strongly-corr_08 | 43801 | 44251 | 43960 | 43694 |
| ch130_n129_bounded-strongly-corr_09 | 43655 | 44963 | 44879 | 44563 |
| ch130_n129_bounded-strongly-corr_10 | 41689 | 42817 | 42787 | 42564 |
| ch130_n129_uncorr_01 | 6499 | 6895 | 6838 | 6741 |
| ch130_n129_uncorr_02 | 7315 | 7513 | 7461 | 7337 |
| ch130_n129_uncorr_03 | 11923 | 12021 | 11989 | 11896 |
| ch130_n129_uncorr_04 | 15154 | 15433 | 15357 | 15209 |
| ch130_n129_uncorr_05 | 17730 | 17890 | 17858 | 17673 |
| ch130_n129_uncorr_06 | 18846 | 18987 | 18941 | 18890 |
| ch130_n129_uncorr_07 | 20169 | 20325 | 20279 | 20153 |
| ch130_n129_uncorr_08 | 21320 | 21535 | 21462 | 21334 |
| ch130_n129_uncorr_09 | 21776 | 22066 | 22010 | 21940 |
| ch130_n129_uncorr_10 | 21882 | 22058 | 22011 | 21879 |
| ch130_n129_uncorr-similar-weights_01 | 4587 | 4615 | 4607 | 4594 |
| ch130_n129_uncorr-similar-weights_02 | 8064 | 8203 | 8179 | 8169 |
| ch130_n129_uncorr-similar-weights_03 | 12495 | 12845 | 12813 | 12700 |
| ch130_n129_uncorr-similar-weights_04 | 14100 | 14430 | 14375 | 14334 |
| ch130_n129_uncorr-similar-weights_05 | 14453 | 14726 | 14610 | 14569 |
| ch130_n129_uncorr-similar-weights_06 | 15746 | 15899 | 15873 | 15832 |

| | | | | |
|---|---|---|---|---|
| ch130_n129_uncorr-similar-weights_07 | 18059 | 18214 | 18186 | 18168 |
| ch130_n129_uncorr-similar-weights_08 | 17373 | 17472 | 17383 | 17060 |
| ch130_n129_uncorr-similar-weights_09 | 18627 | 18662 | 18615 | 18449 |
| ch130_n129_uncorr-similar-weights_10 | 20264 | 20477 | 20450 | 20374 |
| eil51_n50_bounded-strongly-corr_01 | 4269 | 4269 | 4269 | 4269 |
| eil51_n50_bounded-strongly-corr_02 | 5571 | 5571 | 5571 | 5571 |
| eil51_n50_bounded-strongly-corr_03 | 5885 | 5885 | 5885 | 5885 |
| eil51_n50_bounded-strongly-corr_04 | 6310 | 6397 | 6386 | 6286 |
| eil51_n50_bounded-strongly-corr_05 | 4906 | 5138 | 5138 | 5138 |
| eil51_n50_bounded-strongly-corr_06 | 7083 | 7114 | 7113 | 7112 |
| eil51_n50_bounded-strongly-corr_07 | 8240 | 8450 | 8450 | 8450 |
| eil51_n50_bounded-strongly-corr_08 | 7059 | 7342 | 7342 | 7342 |
| eil51_n50_bounded-strongly-corr_09 | 6775 | 6824 | 6820 | 6816 |
| eil51_n50_bounded-strongly-corr_10 | 11000 | 11136 | 11134 | 11110 |
| eil51_n50_uncorr_01 | 2851 | 2871 | 2871 | 2871 |
| eil51_n50_uncorr_02 | 4791 | 4892 | 4892 | 4886 |
| eil51_n50_uncorr_03 | 5404 | 5404 | 5404 | 5404 |
| eil51_n50_uncorr_04 | 3013 | 3013 | 3013 | 3013 |
| eil51_n50_uncorr_05 | 4408 | 4408 | 4408 | 4408 |
| eil51_n50_uncorr_06 | 4440 | 4440 | 4440 | 4440 |
| eil51_n50_uncorr_07 | 4142 | 4165 | 4165 | 4165 |
| eil51_n50_uncorr_08 | 4790 | 4827 | 4827 | 4827 |
| eil51_n50_uncorr_09 | 6122 | 6124 | 6124 | 6124 |
| eil51_n50_uncorr_10 | 6821 | 6906 | 6906 | 6906 |
| eil51_n50_uncorr-similar-weights_01 | 1448 | 1460 | 1460 | 1460 |
| eil51_n50_uncorr-similar-weights_02 | 3769 | 3792 | 3792 | 3792 |
| eil51_n50_uncorr-similar-weights_03 | 4433 | 4515 | 4515 | 4515 |
| eil51_n50_uncorr-similar-weights_04 | 3160 | 3234 | 3234 | 3234 |
| eil51_n50_uncorr-similar-weights_05 | 2134 | 2172 | 2172 | 2172 |
| eil51_n50_uncorr-similar-weights_06 | 2743 | 2743 | 2743 | 2743 |
| eil51_n50_uncorr-similar-weights_07 | 2857 | 2867 | 2867 | 2867 |
| eil51_n50_uncorr-similar-weights_08 | 3452 | 3484 | 3484 | 3484 |
| eil51_n50_uncorr-similar-weights_09 | 4094 | 4200 | 4200 | 4200 |
| eil51_n50_uncorr-similar-weights_10 | 5632 | 5632 | 5632 | 5632 |
| kroA100_n99_bounded-strongly-corr_01 | 4833 | 4976 | 4976 | 4976 |
| kroA100_n99_bounded-strongly-corr_02 | 10167 | 10206 | 10202 | 10183 |
| kroA100_n99_bounded-strongly-corr_03 | 12444 | 12749 | 12648 | 12585 |
| kroA100_n99_bounded-strongly-corr_04 | 15266 | 15585 | 15447 | 15330 |
| kroA100_n99_bounded-strongly-corr_05 | 19926 | 20271 | 20156 | 19958 |
| kroA100_n99_bounded-strongly-corr_06 | 21348 | 22004 | 21902 | 21615 |
| kroA100_n99_bounded-strongly-corr_07 | 22621 | 23497 | 23332 | 22938 |
| kroA100_n99_bounded-strongly-corr_08 | 22135 | 23135 | 23000 | 22485 |
| kroA100_n99_bounded-strongly-corr_09 | 22201 | 23562 | 23355 | 23159 |
| kroA100_n99_bounded-strongly-corr_10 | 22986 | 23956 | 23785 | 23414 |
| kroA100_n99_uncorr_01 | 3891 | 3966 | 3965 | 3953 |
| kroA100_n99_uncorr_02 | 7176 | 7375 | 7375 | 7375 |

| | | | | |
|---|---|---|---|---|
| kroA100_n99_uncorr_03 | 7712 | 7896 | 7896 | 7896 |
| kroA100_n99_uncorr_04 | 10644 | 10760 | 10758 | 10745 |
| kroA100_n99_uncorr_05 | 10838 | 10992 | 10991 | 10978 |
| kroA100_n99_uncorr_06 | 11717 | 11859 | 11857 | 11848 |
| kroA100_n99_uncorr_07 | 12362 | 12421 | 12417 | 12411 |
| kroA100_n99_uncorr_08 | 14780 | 14863 | 14861 | 14856 |
| kroA100_n99_uncorr_09 | 15375 | 15665 | 15664 | 15659 |
| kroA100_n99_uncorr_10 | 16392 | 16510 | 16501 | 16480 |
| kroA100_n99_uncorr-similar-weights_01 | 2286 | 2369 | 2368 | 2363 |
| kroA100_n99_uncorr-similar-weights_02 | 5958 | 6129 | 6129 | 6129 |
| kroA100_n99_uncorr-similar-weights_03 | 8263 | 8578 | 8578 | 8578 |
| kroA100_n99_uncorr-similar-weights_04 | 9303 | 9594 | 9594 | 9594 |
| kroA100_n99_uncorr-similar-weights_05 | 9609 | 9722 | 9722 | 9722 |
| kroA100_n99_uncorr-similar-weights_06 | 11479 | 11818 | 11818 | 11818 |
| kroA100_n99_uncorr-similar-weights_07 | 12953 | 13232 | 13227 | 13181 |
| kroA100_n99_uncorr-similar-weights_08 | 13761 | 13850 | 13846 | 13838 |
| kroA100_n99_uncorr-similar-weights_09 | 15301 | 15377 | 15376 | 15368 |
| kroA100_n99_uncorr-similar-weights_10 | 15999 | 16053 | 16050 | 16046 |
| pr76_n75_bounded-strongly-corr_01 | 2780 | 2797 | 2796 | 2787 |
| pr76_n75_bounded-strongly-corr_02 | 4909 | 4909 | 4909 | 4909 |
| pr76_n75_bounded-strongly-corr_03 | 7291 | 7533 | 7492 | 7322 |
| pr76_n75_bounded-strongly-corr_04 | 6903 | 7237 | 7222 | 7195 |
| pr76_n75_bounded-strongly-corr_05 | 12558 | 12898 | 12887 | 12831 |
| pr76_n75_bounded-strongly-corr_06 | 14659 | 15264 | 15258 | 15202 |
| pr76_n75_bounded-strongly-corr_07 | 16332 | 17381 | 17274 | 16990 |
| pr76_n75_bounded-strongly-corr_08 | 19805 | 20795 | 20793 | 20788 |
| pr76_n75_bounded-strongly-corr_09 | 21309 | 22574 | 22541 | 22519 |
| pr76_n75_bounded-strongly-corr_10 | 24559 | 25772 | 25760 | 25711 |
| pr76_n75_uncorr_01 | 4856 | 5105 | 5105 | 5105 |
| pr76_n75_uncorr_02 | 7602 | 7604 | 7604 | 7604 |
| pr76_n75_uncorr_03 | 9215 | 9293 | 9293 | 9293 |
| pr76_n75_uncorr_04 | 8639 | 8665 | 8664 | 8651 |
| pr76_n75_uncorr_05 | 9208 | 9664 | 9664 | 9664 |
| pr76_n75_uncorr_06 | 9683 | 10238 | 10238 | 10238 |
| pr76_n75_uncorr_07 | 11505 | 11904 | 11904 | 11904 |
| pr76_n75_uncorr_08 | 11401 | 11738 | 11738 | 11738 |
| pr76_n75_uncorr_09 | 11062 | 11431 | 11431 | 11431 |
| pr76_n75_uncorr_10 | 13861 | 14140 | 14140 | 14140 |
| pr76_n75_uncorr-similar-weights_01 | 2564 | 2681 | 2681 | 2681 |
| pr76_n75_uncorr-similar-weights_02 | 4454 | 4723 | 4723 | 4723 |
| pr76_n75_uncorr-similar-weights_03 | 3786 | 4212 | 4212 | 4212 |
| pr76_n75_uncorr-similar-weights_04 | 6044 | 6703 | 6700 | 6692 |
| pr76_n75_uncorr-similar-weights_05 | 6205 | 6759 | 6754 | 6733 |
| pr76_n75_uncorr-similar-weights_06 | 7349 | 7963 | 7956 | 7890 |
| pr76_n75_uncorr-similar-weights_07 | 8455 | 8733 | 8733 | 8733 |
| pr76_n75_uncorr-similar-weights_08 | 10807 | 11037 | 11037 | 11037 |

| | | | | |
|---|---|---|---|---|
| pr76_n75_uncorr-similar-weights_09 | 11037 | 11341 | 11341 | 11341 |
| pr76_n75_uncorr-similar-weights_10 | 12696 | 12873 | 12873 | 12873 |
| rat195_n194_bounded-strongly-corr_01 | 13163 | 13877 | 13749 | 13597 |
| rat195_n194_bounded-strongly-corr_02 | 25752 | 27205 | 27047 | 26874 |
| rat195_n194_bounded-strongly-corr_03 | 28315 | 29875 | 29704 | 29473 |
| rat195_n194_bounded-strongly-corr_04 | 32495 | 34746 | 34442 | 33376 |
| rat195_n194_bounded-strongly-corr_05 | 28210 | 30564 | 30197 | 29859 |
| rat195_n194_bounded-strongly-corr_06 | 32109 | 35511 | 35251 | 34817 |
| rat195_n194_bounded-strongly-corr_07 | 31380 | 35894 | 35548 | 35171 |
| rat195_n194_bounded-strongly-corr_08 | 35876 | 39792 | 39403 | 39064 |
| rat195_n194_bounded-strongly-corr_09 | 40781 | 42868 | 42693 | 42380 |
| rat195_n194_bounded-strongly-corr_10 | 41142 | 44106 | 43695 | 43324 |
| rat195_n194_uncorr_01 | 8288 | 8760 | 8676 | 8596 |
| rat195_n194_uncorr_02 | 15173 | 15822 | 15673 | 15547 |
| rat195_n194_uncorr_03 | 19860 | 21090 | 20852 | 20580 |
| rat195_n194_uncorr_04 | 22275 | 23476 | 23250 | 22899 |
| rat195_n194_uncorr_05 | 24900 | 25972 | 25475 | 25031 |
| rat195_n194_uncorr_06 | 22426 | 22710 | 22571 | 22332 |
| rat195_n194_uncorr_07 | 25550 | 26325 | 25718 | 25521 |
| rat195_n194_uncorr_08 | 24124 | 24878 | 24419 | 24109 |
| rat195_n194_uncorr_09 | 25003 | 25488 | 25202 | 24995 |
| rat195_n194_uncorr_10 | 26329 | 26928 | 26566 | 26295 |
| rat195_n194_uncorr-similar-weights_01 | 4753 | 5310 | 5260 | 5149 |
| rat195_n194_uncorr-similar-weights_02 | 9093 | 10003 | 9843 | 9647 |
| rat195_n194_uncorr-similar-weights_03 | 8978 | 9708 | 9563 | 9427 |
| rat195_n194_uncorr-similar-weights_04 | 11382 | 12184 | 11943 | 11685 |
| rat195_n194_uncorr-similar-weights_05 | 14497 | 15126 | 14983 | 14678 |
| rat195_n194_uncorr-similar-weights_06 | 15075 | 15471 | 15270 | 15128 |
| rat195_n194_uncorr-similar-weights_07 | 18400 | 18944 | 18788 | 18615 |
| rat195_n194_uncorr-similar-weights_08 | 20380 | 20969 | 20757 | 20374 |
| rat195_n194_uncorr-similar-weights_09 | 22842 | 23456 | 23253 | 22953 |
| rat195_n194_uncorr-similar-weights_10 | 24862 | 25332 | 25216 | 25060 |

Table 28: Table showing the results of the 240 studied instances.

| | Benchmark | Best | Average | Worst |
|---|---|---|---|---|
| berlin52_n153_bounded-strongly-corr_01 | 10364 | 10447 | 10445 | 10441 |
| berlin52_n153_bounded-strongly-corr_02 | 18894 | 18949 | 18886 | 18797 |
| berlin52_n153_bounded-strongly-corr_03 | 24276 | 24295 | 24221 | 23953 |
| berlin52_n153_bounded-strongly-corr_04 | 34926 | 34971 | 34901 | 34733 |
| berlin52_n153_bounded-strongly-corr_05 | 37690 | 37713 | 37527 | 37142 |
| berlin52_n153_bounded-strongly-corr_06 | 45292 | 45292 | 45180 | 45163 |
| berlin52_n153_bounded-strongly-corr_07 | 46248 | 46969 | 46969 | 46969 |
| berlin52_n153_bounded-strongly-corr_08 | 51480 | 52380 | 52380 | 52380 |
| berlin52_n153_bounded-strongly-corr_09 | 48913 | 50010 | 50010 | 50010 |

| | | | | |
|---|---|---|---|---|
| berlin52_n153_bounded-strongly-corr_10 | 50221 | 51334 | 51334 | 51334 |
| berlin52_n153_uncorr_01 | 11201 | 11201 | 11201 | 11201 |
| berlin52_n153_uncorr_02 | 12831 | 12839 | 12839 | 12839 |
| berlin52_n153_uncorr_03 | 18800 | 18812 | 18812 | 18812 |
| berlin52_n153_uncorr_04 | 20611 | 20611 | 20611 | 20611 |
| berlin52_n153_uncorr_05 | 20521 | 20521 | 20521 | 20521 |
| berlin52_n153_uncorr_06 | 18634 | 18679 | 18679 | 18679 |
| berlin52_n153_uncorr_07 | 20767 | 20767 | 20767 | 20767 |
| berlin52_n153_uncorr_08 | 22431 | 22431 | 22431 | 22431 |
| berlin52_n153_uncorr_09 | 22100 | 22146 | 22146 | 22146 |
| berlin52_n153_uncorr_10 | 24009 | 24162 | 24162 | 24162 |
| berlin52_n153_uncorr-similar-weights_01 | 5826 | 5832 | 5831 | 5826 |
| berlin52_n153_uncorr-similar-weights_02 | 10197 | 10320 | 10320 | 10320 |
| berlin52_n153_uncorr-similar-weights_03 | 11894 | 12163 | 12163 | 12163 |
| berlin52_n153_uncorr-similar-weights_04 | 12115 | 12565 | 12565 | 12565 |
| berlin52_n153_uncorr-similar-weights_05 | 16625 | 17325 | 17325 | 17325 |
| berlin52_n153_uncorr-similar-weights_06 | 17978 | 19066 | 19066 | 19066 |
| berlin52_n153_uncorr-similar-weights_07 | 20124 | 21489 | 21489 | 21489 |
| berlin52_n153_uncorr-similar-weights_08 | 22911 | 23833 | 23833 | 23833 |
| berlin52_n153_uncorr-similar-weights_09 | 24436 | 25466 | 25466 | 25466 |
| berlin52_n153_uncorr-similar-weights_10 | 27037 | 27626 | 27626 | 27626 |
| berlin52_n255_bounded-strongly-corr_01 | 16367 | 16511 | 16452 | 16401 |
| berlin52_n255_bounded-strongly-corr_02 | 33024 | 33052 | 33026 | 32992 |
| berlin52_n255_bounded-strongly-corr_03 | 50175 | 50289 | 50202 | 49798 |
| berlin52_n255_bounded-strongly-corr_04 | 59965 | 59995 | 59874 | 59631 |
| berlin52_n255_bounded-strongly-corr_05 | 71281 | 71291 | 71028 | 70115 |
| berlin52_n255_bounded-strongly-corr_06 | 86954 | 86944 | 86361 | 86072 |
| berlin52_n255_bounded-strongly-corr_07 | 92856 | 93790 | 93790 | 93790 |
| berlin52_n255_bounded-strongly-corr_08 | 98228 | 99877 | 99877 | 99877 |
| berlin52_n255_bounded-strongly-corr_09 | 97987 | 99675 | 99675 | 99675 |
| berlin52_n255_bounded-strongly-corr_10 | 86780 | 90946 | 90946 | 90946 |
| berlin52_n255_uncorr_01 | 19967 | 20040 | 20040 | 20040 |
| berlin52_n255_uncorr_02 | 25027 | 25247 | 25247 | 25247 |
| berlin52_n255_uncorr_03 | 31163 | 31356 | 31356 | 31356 |
| berlin52_n255_uncorr_04 | 35133 | 35482 | 35482 | 35482 |
| berlin52_n255_uncorr_05 | 36883 | 37324 | 37324 | 37324 |
| berlin52_n255_uncorr_06 | 37259 | 37561 | 37561 | 37561 |
| berlin52_n255_uncorr_07 | 41220 | 41754 | 41754 | 41754 |
| berlin52_n255_uncorr_08 | 40264 | 40566 | 40566 | 40566 |
| berlin52_n255_uncorr_09 | 40539 | 40911 | 40911 | 40911 |
| berlin52_n255_uncorr_10 | 42582 | 43354 | 43354 | 43354 |
| berlin52_n255_uncorr-similar-weights_01 | 10888 | 10939 | 10939 | 10939 |
| berlin52_n255_uncorr-similar-weights_02 | 17226 | 17247 | 17247 | 17247 |
| berlin52_n255_uncorr-similar-weights_03 | 21457 | 21881 | 21881 | 21881 |
| berlin52_n255_uncorr-similar-weights_04 | 24335 | 25116 | 25116 | 25116 |
| berlin52_n255_uncorr-similar-weights_05 | 28571 | 29602 | 29602 | 29602 |

| | | | | |
|---|---|---|---|---|
| berlin52_n255_uncorr-similar-weights_06 | 31825 | 32436 | 32436 | 32436 |
| berlin52_n255_uncorr-similar-weights_07 | 34850 | 35817 | 35817 | 35817 |
| berlin52_n255_uncorr-similar-weights_08 | 40224 | 40735 | 40735 | 40735 |
| berlin52_n255_uncorr-similar-weights_09 | 43247 | 44452 | 44452 | 44452 |
| berlin52_n255_uncorr-similar-weights_10 | 46952 | 47893 | 47893 | 47893 |
| berlin52_n510_bounded-strongly-corr_01 | 31832 | 32352 | 32344 | 32337 |
| berlin52_n510_bounded-strongly-corr_02 | 61259 | 61438 | 61345 | 61254 |
| berlin52_n510_bounded-strongly-corr_03 | 90621 | 90648 | 90457 | 89879 |
| berlin52_n510_bounded-strongly-corr_04 | 113406 | 113425 | 113043 | 112708 |
| berlin52_n510_bounded-strongly-corr_05 | 135417 | 135454 | 134909 | 133022 |
| berlin52_n510_bounded-strongly-corr_06 | 156360 | 156368 | 155671 | 154673 |
| berlin52_n510_bounded-strongly-corr_07 | 168648 | 170350 | 170350 | 170350 |
| berlin52_n510_bounded-strongly-corr_08 | 181775 | 183273 | 183273 | 183273 |
| berlin52_n510_bounded-strongly-corr_09 | 185583 | 187669 | 187669 | 187669 |
| berlin52_n510_bounded-strongly-corr_10 | 174819 | 180308 | 180308 | 180308 |
| berlin52_n510_uncorr_01 | 40122 | 40877 | 40877 | 40877 |
| berlin52_n510_uncorr_02 | 56261 | 57102 | 57102 | 57102 |
| berlin52_n510_uncorr_03 | 61725 | 62478 | 62478 | 62478 |
| berlin52_n510_uncorr_04 | 71536 | 72132 | 72132 | 72132 |
| berlin52_n510_uncorr_05 | 76418 | 77081 | 77081 | 77081 |
| berlin52_n510_uncorr_06 | 80715 | 81393 | 81393 | 81393 |
| berlin52_n510_uncorr_07 | 85398 | 86375 | 86375 | 86375 |
| berlin52_n510_uncorr_08 | 87714 | 88967 | 88967 | 88967 |
| berlin52_n510_uncorr_09 | 88796 | 89727 | 89727 | 89727 |
| berlin52_n510_uncorr_10 | 95381 | 96519 | 96519 | 96519 |
| berlin52_n510_uncorr-similar-weights_01 | 25675 | 25803 | 25803 | 25803 |
| berlin52_n510_uncorr-similar-weights_02 | 38945 | 38945 | 38945 | 38945 |
| berlin52_n510_uncorr-similar-weights_03 | 48460 | 48477 | 48477 | 48477 |
| berlin52_n510_uncorr-similar-weights_04 | 54687 | 54691 | 54691 | 54691 |
| berlin52_n510_uncorr-similar-weights_05 | 61041 | 61744 | 61744 | 61744 |
| berlin52_n510_uncorr-similar-weights_06 | 65548 | 66311 | 66311 | 66311 |
| berlin52_n510_uncorr-similar-weights_07 | 71106 | 71737 | 71737 | 71737 |
| berlin52_n510_uncorr-similar-weights_08 | 77929 | 78985 | 78985 | 78985 |
| berlin52_n510_uncorr-similar-weights_09 | 85698 | 87443 | 87443 | 87443 |
| berlin52_n510_uncorr-similar-weights_10 | 92956 | 94630 | 94630 | 94630 |
| eil51_n150_bounded-strongly-corr_01 | 7169 | 7253 | 7253 | 7253 |
| eil51_n150_bounded-strongly-corr_02 | 13383 | 13447 | 13447 | 13447 |
| eil51_n150_bounded-strongly-corr_03 | 15855 | 15855 | 15767 | 15678 |
| eil51_n150_bounded-strongly-corr_04 | 21641 | 21641 | 21615 | 21379 |
| eil51_n150_bounded-strongly-corr_05 | 22996 | 22996 | 22996 | 22996 |
| eil51_n150_bounded-strongly-corr_06 | 21045 | 21045 | 21045 | 21045 |
| eil51_n150_bounded-strongly-corr_07 | 21226 | 21226 | 21226 | 21226 |
| eil51_n150_bounded-strongly-corr_08 | 20725 | 20725 | 20725 | 20725 |
| eil51_n150_bounded-strongly-corr_09 | 23553 | 23553 | 23553 | 23553 |
| eil51_n150_bounded-strongly-corr_10 | 24359 | 24359 | 24359 | 24359 |
| eil51_n150_uncorr_01 | 6884 | 6884 | 6884 | 6884 |

| | | | | |
|---|---|---|---|---|
| eil51_n150_uncorr_02 | 9331 | 9363 | 9363 | 9363 |
| eil51_n150_uncorr_03 | 11360 | 11360 | 11360 | 11360 |
| eil51_n150_uncorr_04 | 8928 | 9086 | 9086 | 9086 |
| eil51_n150_uncorr_05 | 9734 | 9734 | 9734 | 9734 |
| eil51_n150_uncorr_06 | 11107 | 11135 | 11135 | 11135 |
| eil51_n150_uncorr_07 | 11710 | 11763 | 11763 | 11763 |
| eil51_n150_uncorr_08 | 14415 | 14415 | 14415 | 14415 |
| eil51_n150_uncorr_09 | 17394 | 17394 | 17394 | 17394 |
| eil51_n150_uncorr_10 | 20173 | 20173 | 20173 | 20173 |
| eil51_n150_uncorr-similar-weights_01 | 4275 | 4325 | 4325 | 4325 |
| eil51_n150_uncorr-similar-weights_02 | 7958 | 7958 | 7958 | 7958 |
| eil51_n150_uncorr-similar-weights_03 | 11166 | 11166 | 11166 | 11166 |
| eil51_n150_uncorr-similar-weights_04 | 8082 | 8118 | 8118 | 8118 |
| eil51_n150_uncorr-similar-weights_05 | 8953 | 9103 | 9103 | 9103 |
| eil51_n150_uncorr-similar-weights_06 | 9786 | 9786 | 9786 | 9786 |
| eil51_n150_uncorr-similar-weights_07 | 11637 | 11637 | 11637 | 11637 |
| eil51_n150_uncorr-similar-weights_08 | 12294 | 12294 | 12294 | 12294 |
| eil51_n150_uncorr-similar-weights_09 | 13823 | 13889 | 13889 | 13889 |
| eil51_n150_uncorr-similar-weights_10 | 15736 | 15940 | 15940 | 15940 |
| eil51_n250_bounded-strongly-corr_01 | 11674 | 11734 | 11719 | 11698 |
| eil51_n250_bounded-strongly-corr_02 | 20607 | 20675 | 20675 | 20675 |
| eil51_n250_bounded-strongly-corr_03 | 31651 | 31655 | 31655 | 31655 |
| eil51_n250_bounded-strongly-corr_04 | 34099 | 34101 | 34101 | 34101 |
| eil51_n250_bounded-strongly-corr_05 | 33353 | 33353 | 33297 | 33040 |
| eil51_n250_bounded-strongly-corr_06 | 39486 | 39486 | 39454 | 39158 |
| eil51_n250_bounded-strongly-corr_07 | 38721 | 38721 | 38528 | 38247 |
| eil51_n250_bounded-strongly-corr_08 | 42302 | 42302 | 42302 | 42302 |
| eil51_n250_bounded-strongly-corr_09 | 42440 | 42440 | 42440 | 42440 |
| eil51_n250_bounded-strongly-corr_10 | 39166 | 39166 | 39166 | 39166 |
| eil51_n250_uncorr_01 | 11630 | 11753 | 11753 | 11753 |
| eil51_n250_uncorr_02 | 17652 | 17689 | 17689 | 17689 |
| eil51_n250_uncorr_03 | 15963 | 15963 | 15963 | 15963 |
| eil51_n250_uncorr_04 | 18707 | 18707 | 18707 | 18707 |
| eil51_n250_uncorr_05 | 18815 | 18815 | 18815 | 18815 |
| eil51_n250_uncorr_06 | 20944 | 20944 | 20944 | 20944 |
| eil51_n250_uncorr_07 | 19377 | 19377 | 19377 | 19377 |
| eil51_n250_uncorr_08 | 23593 | 23593 | 23593 | 23593 |
| eil51_n250_uncorr_09 | 26795 | 26795 | 26795 | 26795 |
| eil51_n250_uncorr_10 | 31062 | 31062 | 31062 | 31062 |
| eil51_n250_uncorr-similar-weights_01 | 5655 | 5792 | 5792 | 5792 |
| eil51_n250_uncorr-similar-weights_02 | 10300 | 10300 | 10300 | 10300 |
| eil51_n250_uncorr-similar-weights_03 | 12566 | 12566 | 12566 | 12566 |
| eil51_n250_uncorr-similar-weights_04 | 12061 | 12061 | 12061 | 12061 |
| eil51_n250_uncorr-similar-weights_05 | 14428 | 14428 | 14428 | 14428 |
| eil51_n250_uncorr-similar-weights_06 | 17100 | 17100 | 17100 | 17100 |
| eil51_n250_uncorr-similar-weights_07 | 19973 | 19984 | 19984 | 19984 |

| | | | | |
|---|---|---|---|---|
| eil51_n250_uncorr-similar-weights_08 | 19516 | 19516 | 19516 | 19516 |
| eil51_n250_uncorr-similar-weights_09 | 22623 | 22623 | 22623 | 22623 |
| eil51_n250_uncorr-similar-weights_10 | 27782 | 27806 | 27806 | 27806 |
| eil51_n500_bounded-strongly-corr_01 | 26870 | 26870 | 26862 | 26835 |
| eil51_n500_bounded-strongly-corr_02 | 42950 | 43023 | 42913 | 42605 |
| eil51_n500_bounded-strongly-corr_03 | 64274 | 64276 | 64258 | 64225 |
| eil51_n500_bounded-strongly-corr_04 | 70133 | 70133 | 70090 | 69991 |
| eil51_n500_bounded-strongly-corr_05 | 77597 | 77605 | 77605 | 77605 |
| eil51_n500_bounded-strongly-corr_06 | 80925 | 80925 | 80925 | 80925 |
| eil51_n500_bounded-strongly-corr_07 | 83059 | 83066 | 83065 | 83059 |
| eil51_n500_bounded-strongly-corr_08 | 82008 | 82008 | 82008 | 82008 |
| eil51_n500_bounded-strongly-corr_09 | 81822 | 82004 | 81896 | 81647 |
| eil51_n500_bounded-strongly-corr_10 | 79770 | 80039 | 80039 | 80039 |
| eil51_n500_uncorr_01 | 22722 | 23040 | 23040 | 23040 |
| eil51_n500_uncorr_02 | 34045 | 34045 | 34045 | 34045 |
| eil51_n500_uncorr_03 | 42481 | 42494 | 42494 | 42494 |
| eil51_n500_uncorr_04 | 42388 | 42411 | 42411 | 42411 |
| eil51_n500_uncorr_05 | 46899 | 46899 | 46899 | 46899 |
| eil51_n500_uncorr_06 | 46710 | 46829 | 46829 | 46829 |
| eil51_n500_uncorr_07 | 45380 | 45380 | 45380 | 45380 |
| eil51_n500_uncorr_08 | 51236 | 51236 | 51236 | 51236 |
| eil51_n500_uncorr_09 | 56653 | 56653 | 56653 | 56653 |
| eil51_n500_uncorr_10 | 65075 | 65075 | 65075 | 65075 |
| eil51_n500_uncorr-similar-weights_01 | 13398 | 13407 | 13407 | 13407 |
| eil51_n500_uncorr-similar-weights_02 | 20710 | 20710 | 20667 | 20495 |
| eil51_n500_uncorr-similar-weights_03 | 23372 | 23372 | 23372 | 23372 |
| eil51_n500_uncorr-similar-weights_04 | 22658 | 22789 | 22789 | 22789 |
| eil51_n500_uncorr-similar-weights_05 | 29666 | 29679 | 29679 | 29679 |
| eil51_n500_uncorr-similar-weights_06 | 33347 | 33347 | 33347 | 33347 |
| eil51_n500_uncorr-similar-weights_07 | 38734 | 38734 | 38734 | 38734 |
| eil51_n500_uncorr-similar-weights_08 | 42567 | 42567 | 42567 | 42567 |
| eil51_n500_uncorr-similar-weights_09 | 48774 | 48774 | 48774 | 48774 |
| eil51_n500_uncorr-similar-weights_10 | 55972 | 55972 | 55972 | 55972 |
| pr76_n225_bounded-strongly-corr_01 | 15366 | 15703 | 15677 | 15614 |
| pr76_n225_bounded-strongly-corr_02 | 21542 | 21729 | 21697 | 21648 |
| pr76_n225_bounded-strongly-corr_03 | 27422 | 27630 | 27606 | 27578 |
| pr76_n225_bounded-strongly-corr_04 | 34894 | 35572 | 35534 | 35464 |
| pr76_n225_bounded-strongly-corr_05 | 44469 | 46044 | 45992 | 45787 |
| pr76_n225_bounded-strongly-corr_06 | 46162 | 48722 | 48722 | 48722 |
| pr76_n225_bounded-strongly-corr_07 | 57890 | 61958 | 61958 | 61958 |
| pr76_n225_bounded-strongly-corr_08 | 59366 | 64072 | 64072 | 64072 |
| pr76_n225_bounded-strongly-corr_09 | 65000 | 69948 | 69948 | 69948 |
| pr76_n225_bounded-strongly-corr_10 | 69642 | 72785 | 72785 | 72785 |
| pr76_n225_uncorr_01 | 18018 | 18558 | 18558 | 18558 |
| pr76_n225_uncorr_02 | 14059 | 14317 | 14203 | 13747 |
| pr76_n225_uncorr_03 | 17909 | 18507 | 18507 | 18507 |

| | | | | |
|---|---|---|---|---|
| pr76_n225_uncorr_04 | 19486 | 19749 | 19730 | 19564 |
| pr76_n225_uncorr_05 | 21151 | 21336 | 21330 | 21315 |
| pr76_n225_uncorr_06 | 25365 | 25799 | 25797 | 25774 |
| pr76_n225_uncorr_07 | 27652 | 28096 | 28096 | 28096 |
| pr76_n225_uncorr_08 | 32456 | 32833 | 32781 | 32659 |
| pr76_n225_uncorr_09 | 32855 | 33539 | 33530 | 33450 |
| pr76_n225_uncorr_10 | 35183 | 35436 | 35419 | 35401 |
| pr76_n225_uncorr-similar-weights_01 | 7971 | 8138 | 8137 | 8137 |
| pr76_n225_uncorr-similar-weights_02 | 11254 | 11708 | 11708 | 11708 |
| pr76_n225_uncorr-similar-weights_03 | 15338 | 16084 | 16084 | 16084 |
| pr76_n225_uncorr-similar-weights_04 | 16050 | 16830 | 16772 | 16533 |
| pr76_n225_uncorr-similar-weights_05 | 17305 | 17900 | 17890 | 17803 |
| pr76_n225_uncorr-similar-weights_06 | 21579 | 21723 | 21721 | 21701 |
| pr76_n225_uncorr-similar-weights_07 | 25921 | 26278 | 26278 | 26278 |
| pr76_n225_uncorr-similar-weights_08 | 27194 | 28000 | 28000 | 28000 |
| pr76_n225_uncorr-similar-weights_09 | 31673 | 32052 | 32050 | 32034 |
| pr76_n225_uncorr-similar-weights_10 | 34309 | 35182 | 35172 | 35078 |
| pr76_n375_bounded-strongly-corr_01 | 24306 | 24656 | 24581 | 24424 |
| pr76_n375_bounded-strongly-corr_02 | 35589 | 35967 | 35940 | 35889 |
| pr76_n375_bounded-strongly-corr_03 | 46870 | 47564 | 47559 | 47546 |
| pr76_n375_bounded-strongly-corr_04 | 57189 | 58371 | 58355 | 58307 |
| pr76_n375_bounded-strongly-corr_05 | 74211 | 75227 | 75199 | 75188 |
| pr76_n375_bounded-strongly-corr_06 | 86303 | 89089 | 89054 | 89003 |
| pr76_n375_bounded-strongly-corr_07 | 96614 | 99427 | 99287 | 99077 |
| pr76_n375_bounded-strongly-corr_08 | 102211 | 102682 | 102682 | 102682 |
| pr76_n375_bounded-strongly-corr_09 | 108360 | 111114 | 111114 | 111114 |
| pr76_n375_bounded-strongly-corr_10 | 104239 | 109152 | 109152 | 109152 |
| pr76_n375_uncorr_01 | 25075 | 25980 | 25970 | 25873 |
| pr76_n375_uncorr_02 | 29563 | 30092 | 30084 | 30067 |
| pr76_n375_uncorr_03 | 34047 | 35044 | 35036 | 35017 |
| pr76_n375_uncorr_04 | 37538 | 39062 | 39062 | 39062 |
| pr76_n375_uncorr_05 | 38051 | 38833 | 38792 | 38529 |
| pr76_n375_uncorr_06 | 43834 | 44311 | 44311 | 44311 |
| pr76_n375_uncorr_07 | 49318 | 50278 | 50278 | 50278 |
| pr76_n375_uncorr_08 | 52215 | 53436 | 53436 | 53436 |
| pr76_n375_uncorr_09 | 56362 | 57215 | 57215 | 57215 |
| pr76_n375_uncorr_10 | 58827 | 60243 | 60243 | 60243 |
| pr76_n375_uncorr-similar-weights_01 | 14138 | 14351 | 14351 | 14351 |
| pr76_n375_uncorr-similar-weights_02 | 23487 | 24548 | 24548 | 24548 |
| pr76_n375_uncorr-similar-weights_03 | 27485 | 28761 | 28761 | 28761 |
| pr76_n375_uncorr-similar-weights_04 | 27196 | 27882 | 27882 | 27882 |
| pr76_n375_uncorr-similar-weights_05 | 31269 | 31580 | 31580 | 31580 |
| pr76_n375_uncorr-similar-weights_06 | 33212 | 34393 | 34393 | 34393 |
| pr76_n375_uncorr-similar-weights_07 | 38352 | 40479 | 40479 | 40479 |
| pr76_n375_uncorr-similar-weights_08 | 45350 | 46661 | 46661 | 46661 |
| pr76_n375_uncorr-similar-weights_09 | 52059 | 52851 | 52851 | 52851 |

| | | | | |
|---|---|---|---|---|
| pr76_n375_uncorr-similar-weights_10 | 56880 | 57648 | 57648 | 57648 |
| pr76_n750_bounded-strongly-corr_01 | 45586 | 46435 | 46357 | 46301 |
| pr76_n750_bounded-strongly-corr_02 | 74809 | 75065 | 74904 | 74220 |
| pr76_n750_bounded-strongly-corr_03 | 105096 | 106000 | 105578 | 105096 |
| pr76_n750_bounded-strongly-corr_04 | 123691 | 125571 | 125571 | 125571 |
| pr76_n750_bounded-strongly-corr_05 | 145783 | 148150 | 148120 | 147846 |
| pr76_n750_bounded-strongly-corr_06 | 161895 | 164576 | 164316 | 163205 |
| pr76_n750_bounded-strongly-corr_07 | 178660 | 182791 | 182791 | 182791 |
| pr76_n750_bounded-strongly-corr_08 | 185533 | 188935 | 188935 | 188935 |
| pr76_n750_bounded-strongly-corr_09 | 192163 | 198073 | 198073 | 198073 |
| pr76_n750_bounded-strongly-corr_10 | 191703 | 198683 | 198683 | 198683 |
| pr76_n750_uncorr_01 | 42412 | 43981 | 43981 | 43981 |
| pr76_n750_uncorr_02 | 64525 | 65043 | 65043 | 65043 |
| pr76_n750_uncorr_03 | 73314 | 74299 | 74299 | 74299 |
| pr76_n750_uncorr_04 | 83825 | 84284 | 84284 | 84284 |
| pr76_n750_uncorr_05 | 88062 | 89054 | 89054 | 89054 |
| pr76_n750_uncorr_06 | 95701 | 97631 | 97631 | 97631 |
| pr76_n750_uncorr_07 | 105461 | 106757 | 106757 | 106757 |
| pr76_n750_uncorr_08 | 115087 | 116661 | 116661 | 116661 |
| pr76_n750_uncorr_09 | 119245 | 122762 | 122762 | 122762 |
| pr76_n750_uncorr_10 | 126317 | 128846 | 128846 | 128846 |
| pr76_n750_uncorr-similar-weights_01 | 28602 | 28813 | 28738 | 28715 |
| pr76_n750_uncorr-similar-weights_02 | 43265 | 43691 | 43682 | 43668 |
| pr76_n750_uncorr-similar-weights_03 | 53943 | 54208 | 54208 | 54208 |
| pr76_n750_uncorr-similar-weights_04 | 59383 | 61126 | 61126 | 61126 |
| pr76_n750_uncorr-similar-weights_05 | 70169 | 71931 | 71931 | 71931 |
| pr76_n750_uncorr-similar-weights_06 | 77243 | 80610 | 80610 | 80610 |
| pr76_n750_uncorr-similar-weights_07 | 90223 | 92090 | 92090 | 92090 |
| pr76_n750_uncorr-similar-weights_08 | 98333 | 102098 | 102098 | 102098 |
| pr76_n750_uncorr-similar-weights_09 | 109815 | 110536 | 110536 | 110536 |
| pr76_n750_uncorr-similar-weights_10 | 121977 | 122718 | 122718 | 122718 |

Table 29: Table showing the results of the extra 270 instances

Table 27: Table showing the time in minutes it takes for 2500 local optima to complete with a steady state genetic algorithm.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| eil51 | BSC | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.3 | 0.3 | 0.2 |
| | USW | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| | U | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| berlin52 | BSC | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| | USW | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| | U | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| pr76 | BSC | 0.4 | 0.5 | 0.5 | 0.6 | 0.6 | 0.6 | 0.7 | 0.7 | 0.6 | 0.6 |
| | USW | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| | U | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.4 |
| kroA100 | BSC | 0.8 | 1 | 1 | 1.1 | 1.4 | 1.2 | 1.4 | 1.3 | 1.3 | 1.2 |
| | USW | 0.8 | 0.8 | 0.9 | 1 | 1 | 1 | 1 | 1 | 0.8 | 0.8 |
| | U | 0.6 | 0.7 | 0.8 | 0.8 | 0.9 | 0.9 | 1 | 0.9 | 0.9 | 0.8 |
| bier127 | BSC | 2.1 | 2.1 | 1.9 | 1.9 | 1.8 | 1.9 | 1.9 | 2 | 1.9 | 1.9 |
| | USW | 2.2 | 2.1 | 2 | 1.8 | 1.9 | 1.7 | 1.7 | 1.6 | 1.7 | 1.5 |
| | U | 2 | 1.6 | 2 | 1.9 | 1.8 | 1.9 | 1.7 | 1.5 | 1.7 | 1.6 |
| ch130 | BSC | 1.7 | 1.5 | 1.9 | 2.1 | 2.3 | 2.6 | 2.8 | 2.5 | 2.3 | 2.2 |
| | USW | 1.7 | 1.6 | 1.7 | 1.9 | 1.8 | 2 | 1.6 | 1.7 | 1.6 | 1.6 |
| | U | 1.6 | 1.6 | 1.7 | 1.7 | 1.8 | 1.7 | 1.6 | 1.5 | 1.5 | 1.6 |
| rat195 | BSC | 5.9 | 5.8 | 6.2 | 8.8 | 9 | 9.6 | 9.4 | 8.2 | 8.3 | 8.2 |
| | USW | 5.3 | 5.7 | 6 | 6.2 | 6.4 | 6.5 | 6.8 | 6.4 | 5.7 | 5.4 |
| | U | 4.7 | 5.1 | 5.9 | 5.9 | 6.5 | 6.2 | 6.3 | 5.9 | 6.1 | 5.4 |
| a280 | BSC | 14.9 | 16 | 19.6 | 22.5 | 21.9 | 21.8 | 20.9 | 22.1 | 19.1 | 22.7 |
| | USW | 15.2 | 15.1 | 15.3 | 17.3 | 15.8 | 16.2 | 15.7 | 14.9 | 14.1 | 14.5 |
| | U | 13.7 | 14 | 14 | 14.6 | 16.6 | 16.3 | 16.3 | 15.4 | 14.3 | 14.9 |

# References


[1] David L Applegate et al. *The traveling salesman problem: a computational study*. Princeton university press, 2011.

[2] David Applegate, William Cook, and André Rohe. "Chained Lin-Kernighan for large traveling salesman problems". In: *INFORMS Journal on Computing* 15.1 (2003), pp. 82–92.

[3] Rodolfo Pereira Araujo et al. "A novel List-Constrained Randomized VND approach in GPU for the Traveling Thief Problem". In: *Electronic Notes in Discrete Mathematics* 66 (2018), pp. 183–190.

[4] James C Bean. "Genetic algorithms and random keys for sequencing and optimization". In: *ORSA journal on computing* 6.2 (1994), pp. 154–160.

[5] Kenneth D Boese, Andrew B Kahng, and Sudhakar Muddu. "A new adaptive multi-start technique for combinatorial global optimizations". In: *Operations Research Letters* 16.2 (1994), pp. 101–113.

[6] Kenneth D Boese, Andrew B Kahng, and Sudhakar Muddu. "On the big valley and adaptive multi-start for discrete global optimizations". In: *Comput. Sci. Dept., Univ. California, Los Angeles, Tech. Rep. TR-930* 15 (1993).

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

[8] Mohammad Reza Bonyadi et al. "Socially inspired algorithms for the travelling thief problem". In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2014, pp. 421–428.

[9] Nicos Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem*. Tech. rep. Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[10] William Cook. *In pursuit of the traveling salesman: mathematics at the limits of computation*. Princeton University Press, 2012.

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

[12] Lawrence Davis. "Applying adaptive algorithms to epistatic domains." In: *IJCAI*. Vol. 85. 1985, pp. 162–164.

[13] Mohamed El Yafrani and Belaid Ahiod. "A local search based approach for solving the Travelling Thief Problem: The pros and cons". In: *Applied Soft Computing* 52 (2017), pp. 795–804.

[14] Mohamed El Yafrani and Belaid Ahiod. "Population-based vs. single-solution heuristics for the travelling thief problem". In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM. 2016, pp. 317–324.

[15] Mohamed El Yafrani et al. “A hyperheuristic approach based on low-level heuristics for the travelling thief problem”. In: *Genetic Programming and Evolvable Machines* (), pp. 1–30.

[16] Hayden Faulkner et al. “Approximate approaches to the traveling thief problem”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 385–392.

[17] David E Goldberg, Robert Lingle, et al. “Alleles, loci, and the traveling salesman problem”. In: *Proceedings of an international conference on genetic algorithms and their applications*. Vol. 154. Lawrence Erlbaum, Hillsdale, NJ. 1985, pp. 154–159.

[18] Gregory Gutin, Anders Yeo, and Alexey Zverovich. “Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP”. In: *Discrete Applied Mathematics* 117.1 (2002), pp. 81–86.

[19] Michael Hahsler and Kurt Hornik. “TSP-Infrastructure for the traveling salesperson problem”. In: *Journal of Statistical Software* 23.2 (2007), pp. 1–21.

[20] Doug R Hains, L Darrell Whitley, and Adele E Howe. “Revisiting the big valley search space structure in the TSP”. In: *Journal of the Operational Research Society* 62.2 (2011), pp. 305–312.

[21] Michael Held and Richard M Karp. “A dynamic programming approach to sequencing problems”. In: *Journal of the Society for Industrial and Applied Mathematics* 10.1 (1962), pp. 196–210.

[22] Keld Helsgaun. “An effective implementation of the Lin–Kernighan traveling salesman heuristic”. In: *European Journal of Operational Research* 126.1 (2000), pp. 106–130.

[23] David S Johnson and Lyle A McGeoch. “The traveling salesman problem: A case study in local optimization”. In: *Local search in combinatorial optimization* 1 (1997), pp. 215–310.

[24] Terry Jones et al. “Evolutionary algorithms, fitness landscapes and search”. PhD thesis. University of New Mexico Albuquerque, NM, 1995.

[25] Terry Jones and Stephanie Forrest. “Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms”. In: (1995).

[26] Terry Jones, Gregory JE Rawlins, et al. “Reverse HillclimbingGenetic Algorithms and the Busy Beaver Problem.” In: *ICGA*. 1993, pp. 70–75.

[27] Pedro Larranaga et al. “Genetic algorithms for the travelling salesman problem: A review of representations and operators”. In: *Artificial Intelligence Review* 13.2 (1999), pp. 129–170.

[28] Shen Lin and Brian W Kernighan. “An effective heuristic algorithm for the traveling-salesman problem”. In: *Operations research* 21.2 (1973), pp. 498–516.

[29] Nuno Lourenço, Francisco B Pereira, and Ernesto Costa. "An evolutionary approach to the full optimization of the traveling thief problem". In: *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer. 2016, pp. 34–45.

[30] Bernard Manderick. "The genetic algorithm and the structure of the fitness landscape". In: *Proc. 4th International Conference on Genetic Algorithms*. 1991, pp. 143–150.

[31] Silvano Martello, David Pisinger, and Paolo Toth. "Dynamic programming and strong bounds for the 0-1 knapsack problem". In: *Management Science* 45.3 (1999), pp. 414–424.

[32] Silvano Martello, David Pisinger, and Paolo Toth. "New trends in exact algorithms for the 0–1 knapsack problem". In: *European Journal of Operational Research* 123.2 (2000), pp. 325–332.

[33] Marcella SR Martins et al. "HSEDA: A Heuristic Selection Approach Based on Estimation of Distribution Algorithm for the Travelling Thief Problem". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, p. 8.

[34] Keith Mathias and Darrell Whitley. "Genetic operators, the fitness landscape and the traveling salesman problem". In: *PPSN*. 1992, pp. 221–230.

[35] Dirk C Mattfeld, Christian Bierwirth, and Herbert Kopfer. "A search space analysis of the job shop scheduling problem". In: *Annals of Operations Research* 86 (1999), pp. 441–453.

[36] Yi Mei, Xiaodong Li, and Xin Yao. "Improving efficiency of heuristics for the large scale traveling thief problem". In: *Asia-Pacific Conference on Simulated Evolution and Learning*. Springer. 2014, pp. 631–643.

[37] Yi Mei, Xiaodong Li, and Xin Yao. "On investigation of interdependence between sub-problems of the travelling thief problem". In: *Soft Computing* 20.1 (2016), pp. 157–172.

[38] Yi Mei et al. "Heuristic evolution with genetic programming for traveling thief problem". In: *Evolutionary Computation (CEC), 2015 IEEE Congress on*. IEEE. 2015, pp. 2753–2760.

[39] Peter Merz. "Advanced fitness landscape analysis and the performance of memetic algorithms". In: *Evolutionary Computation* 12.3 (2004), pp. 303–325.

[40] Peter Merz. "Memetic algorithms for combinatorial optimization problems: fitness landscapes and effective search strategies". In: (2006).

[41] Mahdi Moeini, Daniel Schermer, and Oliver Wendt. "A Hybrid Evolutionary Approach for Solving the Traveling Thief Problem". In: *International Conference on Computational Science and Its Applications*. Springer. 2017, pp. 652–668.

[42] Heinz Mühlenbein, Martina Gorges-Schleuter, and Ottmar Krämer. "Evolution algorithms in combinatorial optimization". In: *Parallel Computing* 7.1 (1988), pp. 65–85.

[43] Yuichi Nagata. "New EAX crossover for large TSP instances". In: Springer.

[44] Yuichi Nagata and Shigenobu Kobayashi. "A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem". In: *INFORMS Journal on Computing* 25.2 (2013), pp. 346–363.

[45] Frank Neumann et al. "A Fully Polynomial Time Approximation Scheme for Packing While Traveling". In: *arXiv preprint arXiv:1702.05217* (2017).

[46] Gabriela Ochoa and Nadarajen Veerapen. "Additional dimensions to the study of funnels in combinatorial landscapes". In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM. 2016, pp. 373–380.

[47] Gabriela Ochoa and Nadarajen Veerapen. "Deconstructing the big valley search space hypothesis". In: *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer. 2016, pp. 58–73.

[48] Gabriela Ochoa et al. "The multi-funnel structure of TSP fitness landscapes: a visual exploration". In: *International Conference on Artificial Evolution (Evolution Artificielle)*. Springer. 2015, pp. 1–13.

[49] IM Oliver, DJd Smith, and John RC Holland. "Study of permutation crossover operators on the traveling salesman problem". In: *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*. Hillsdale, NJ: L. Erlhaum Associates, 1987. 1987.

[50] David Pisinger. "A minimal algorithm for the 0-1 knapsack problem". In: *Operations Research* 45.5 (1997), pp. 758–767.

[51] David Pisinger. "An expanding-core algorithm for the exact 0–1 knapsack problem". In: *European Journal of Operational Research* 87.1 (1995), pp. 175–187.

[52] David Pisinger. "Where are the hard knapsack problems?" In: *Computers & Operations Research* 32.9 (2005), pp. 2271–2284.

[53] Erik Pitzer and Michael Affenzeller. "A Comprehensive Survey on Fitness Landscape Analysis." In: *Recent Advances in Intelligent Engineering Systems* 378 (2012), pp. 161–191.

[54] Sergey Polyakovskiy and Frank Neumann. "Packing while traveling: Mixed integer programming for a class of nonlinear knapsack problems". In: *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer. 2015, pp. 332–346.

[55] Sergey Polyakovskiy and Frank Neumann. “The packing while traveling problem”. In: *European Journal of Operational Research* 258.2 (2017), pp. 424–439.

[56] Sergey Polyakovskiy et al. “A comprehensive benchmark set and heuristics for the traveling thief problem”. In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2014, pp. 477–484.

[57] Gerhard Reinelt. “TSPLIB—A traveling salesman problem library”. In: *ORSA journal on computing* 3.4 (1991), pp. 376–384.

[58] Danilo Sanches, Darrell Whitley, and Renato Tinós. “Building a better heuristic for the traveling salesman problem: Combining edge assembly crossover and partition crossover”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2017, pp. 329–336.

[59] Steven Skiena. “Who is interested in algorithms and why?: lessons from the stony brook algorithms repository”. In: *ACM SIGACT News* 30.3 (1999), pp. 65–74.

[60] Timothy Starkweather et al. “A Comparison of Genetic Sequencing Operators.” In: *ICGA*. 1991, pp. 69–76.

[61] Mohammad-H Tayarani-N and Adam Prügel-Bennett. “An analysis of the fitness landscape of travelling salesman problem”. In: *Evolutionary computation* 24.2 (2016), pp. 347–384.

[62] Daniel KS Vieira et al. “A Genetic Algorithm for Multi-component Optimization Problems: The Case of the Travelling Thief Problem”. In: *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer. 2017, pp. 18–29.

[63] Markus Wagner. “Stealing items more efficiently with ants: a swarm intelligence approach to the travelling thief problem”. In: *International Conference on Swarm Intelligence*. Springer. 2016, pp. 273–281.

[64] Markus Wagner et al. “A case study of algorithm selection for the traveling thief problem”. In: *Journal of Heuristics* (2017), pp. 1–26.

[65] Edward Weinberger. “Correlated and uncorrelated fitness landscapes and how to tell the difference”. In: *Biological cybernetics* 63.5 (1990), pp. 325–336.

[66] Darrell Whitley, Doug Hains, and Adele Howe. “A hybrid genetic algorithm for the traveling salesman problem using generalized partition crossover”. In: *Parallel Problem Solving from Nature, PPSN XI* (2010), pp. 566–575.

[67] Darrell Whitley, Doug Hains, and Adele Howe. “Tunneling between optima: partition crossover for the traveling salesman problem”. In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM. 2009, pp. 915–922.

[68] Darrell Whitley and Jonathan Rowe. "Single-Funnel and Multi-funnel Landscapes and Subthreshold-Seeking Behavior". In: *Theory and Principled Methods for the Design of Metaheuristics*. Springer, 2014, pp. 63–84.

[69] L Darrell Whitley and Timothy Starkweather. "Scheduling problems and traveling salesmen: The genetic edge recombination operator". In:

[70] Junhua Wu, Sergey Polyakovskiy, and Frank Neumann. "On the impact of the renting rate for the unconstrained nonlinear knapsack problem". In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM. 2016, pp. 413–419.

[71] Junhua Wu et al. "Exact Approaches for the Travelling Thief Problem". In: *Asia-Pacific Conference on Simulated Evolution and Learning*. Springer. 2017, pp. 110–121.