ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

The Netherlands

July 7, 2019

**Thesis**

# Lin - Kernighan algorithm for the Travelling Salesman Problem and an application to the Travelling Thief Problem

**Wout Konings**

433246

A thesis presented for the bachelor degrees of

Econometrics and Operations Research

&

Economics and Business Economics

| Supervised by: | Second reader: |
|---|---|
| T. Breugem MSc | R. Hoogervorst MSc |

**Abstract**

The Lin-Kernighan algorithm is one of the best performing heuristics for the Travelling Salesman Problem (TSP). In this thesis the literature on the TSP is discussed and a thorough explanation of the Lin-Kernighan algorithm and all state-of-the-art extensions are given. We implement the original algorithm by Lin and Kernighan (1973) and perform an experimental analysis on various TSP*lib* instances to test the algorithm's performace. We find runtimes to grow with rate approximately equal to $n^{2.6}$ and the performance worse than reported in the literature.

Additionally, we look at a relatively new benchmark problem: the Travelling Thief Problem (TTP). The TTP tries to capture the interdependency between subproblems present in many real-life problems in a benchmark problem by combining the TSP with the Knapsack Problem. An overview of the current solvers is given and a new construction algorithm, the Lin-Kernighan Modified Distance (LKMD) algorithm, is proposed. While most current state-of-the-art construction algorithms first pick a tour and then try to find a packing plan, the LKMD does the opposite: it first constructs a packing plan, which is afterwards used to find a good tour. The algorithm is tested on the 30 *eil51* TTP instances. While overall solution quality of the algorithm is good, the running times are 60 times larger than the other construction algorithms. Still, the algorithm may serve as inspiration for further attempts to find heuristics that exploit the interdependency between the two subproblems.

# Contents

## List of Tables

## List of Figures

## 1 Introduction

The Travelling Salesman Problem (TSP) is the problem of finding the shortest tour among $n$ cities, where every city is visited exactly once. The TSP is an extensively studied problem within the fields of combinatorics and operations research. After appearing in the 18th century it was firstly formalised by Karl Menger in 1932 as the *Botenproblem* (Messenger Problem) and has seen a considerable amount of academic attention since. It has probably been the most extensively studied combinatorial problem (Applegate et al., 2006). In a world where optimisation and the reduction of operating costs is ever more important, the TSP remains a relevant problem to be studied.

Its applications are wide, from vehicle routing and various scheduling problems to computer wiring and the drilling of computer boards (Helsgaun, 2000), x-ray crystallography (Bland and Shallcross, 1989), 3D printing and job sequencing. The TSP is also prevalent in warehouse order picking (Ratliff and Rosenthal, 1983). The TSP has become important yet again with upcoming air transportation in the field of parcel delivery and natural disaster emergency handling, which both require fast and good solutions (Wu et al., 2019).

The TSP belongs to the class of NP-complete problems (provided $P \neq NP$; Karp, 1972; Laporte, 1992). This means that finding an exact solution cannot be done in polynomial time and all NP problems can be reduced to the TSP. This has a consequence that larger problem instances cannot be solved in reasonable computation times. For a problem with $n$ cities there are $\frac{(n-1)!}{2}$ possible tours. Thus, for a 10-city problem, there are more than $10^5$ possible tours, and for a 100-city problem more than $10^{150}$, whereas the number of particles in the universe is estimated to be approximately equal to $10^{80}$ (Heile, 2017) [1]. Therefore one often has to recourse to heuristic methods.

One of these heuristics is the Lin-Kernighan (LK) heuristic proposed by Lin and Kernighan (1973), which is still regarded as one of the most sophisticated heuristics for solving the TSP (Chauhan et al., 2012). The heuristic is based on a local search, where many randomly generated tours are sought to be improved to arrive at local optima. The idea is that when a sufficient amount of random starting solutions are investigated together along with a good improvement method, the global optimum will at some point be found amongst the set of found local optima.

A popular local search method is the $k$-opt search, where $k$ edges are removed from the tour and replaced by $k$ edges previously not included in the tour, resulting in a better solution. The higher $k$ is, the higher the chance that the newly constructed tour will be optimal. With 2-opt and 3-opt algorithms 2 or 3 edges are swapped iteratively until no further improvement can be made. However, it is difficult to know in advance what the best amount of edges to switch is, and computation times rise considerably with an increase in $k$. Lin and Kernighan (1973) generalised and revolutionised this approach, by iteratively looking for a $k$, the number of edges to swap, resulting in a variable $k$-opt algorithm. Lin and Kernighan report a running time of $\mathcal{O}(n^{2.2})$ and the Johnson (1990) implementation ordinarily generates solutions less than 2% from the optimal value.

[1] retrieved from www.huffpost.com

Since its original publication, the LK heuristic has been optimised and has been used to solve very large TSP instances. Additionally, it has been altered and implemented into various variations of the TSP. In this research we implement the LK heuristic and test its performance on well known problems in the TSP*lib*, first constructed by Reinelt (1991) and expanded since. Of all these problems the optimal solutions are known.

Some researchers have critiqued modern day operation research for is still focused on 'old' benchmark problems (such as the TSP), many of which do not capture characteristics of real life problems. By not investigating these characteristics, modern day heuristics may not be good enough for dealing with real life problems.

Therefore, Bonyadi et al. (2013) have tried to capture one of those characteristics, that is interdependence between subproblems, by introducing a new problem: the Travelling Thief Problem (TTP). The Travelling Thief Problem is a combination of the Travelling Salesman Problem and the Knapsack Problem. A thief has to travel along all cities and in those cities, items with a certain value and a certain weight are located. The thief pays a certain amount of rent per time unit for the use of the knapsack and aims to maximise the profit of his trip. Picking up more items does however make his knapsack heavier, leading to a slower travel speed and thus a longer trip duration and a higher rent. We can see that the two problems are interrelated, as changing the items to pick could change the best route to choose and vice versa. Additionally, the problems are indecomposable, which means the optimal TSP route need not be the optimal route in the TTP problem and vice versa the optimal picking strategy in the seperate knapsack problem also need not be the optimal picking strategy in the TTP problem. This will be explained in greater detail in section **??**.

The Lin-Kernighan heuristic may be suitable for a construction algorithm for the Travelling Thief Problem, especially when it takes into account the interdependecy of the two problems (Mei et al., 2016). In this research, we propose a new construction algorithm for the TTP: the Lin-Kernighan Modified Distance (LKMD) algorithm. We test the algorithms performance against the current state-of-the-art construction algorithms on the 30 *eil51* TTP instances.

In section 2 we discuss the relevant literature on the TSP so far. The LK algorithm is concisely explained in section 3 along with the relevant literature on the LK algorithm in section 3.1. In section 4 we give a thorough explanation of the TTP, review the solution methods currently available, and explain the LKMD algorithm. Finally in section 5 the experimental results are discussed and the thesis is concluded in section 6.

## 2 Literature review

### 2.1 Travelling Salesman Problem

The Travelling Salesman Problem is the problem of finding the route with the lowest cost, such that every city is visited once. The problem can be modelled by having a complete undirected graph $G = \{V, E\}$, with $V$ the set of nodes representing the cities and $E$ the set of edges where edge $e_{ij}$ represents the shortest route from city $i$ to city $j$. Each edge has a corresponding cost $|e_{ij}| = c_{ij}$. In the case of Euclidean distances, the following hold:

- $c_{ij} = c_{ji}$, the problem is symmetric.
- $c_{ik} \leq c_{ij} + c_{jk}$, the triangular inequality holds. This means that the cost of edge $e_{ij}$ is the shortest and one cannot have a smaller cost by travelling via another edge.

The Travelling Salesman Problem can be modelled as an Integer Program. We have decision variable $x_{ij} = 1$, if edge $e_{ij}$ is selected, 0 otherwise. The formulation then is as follows:

$$min \sum_{e_{ij} \in E} x_{ij} c_{ij} \tag{1}$$

subject to:

$$\sum_{j \in V} x_{ij} = 2, \text{ for every } i \in V \tag{2}$$

$$\sum_{i,j \in V, i \neq j} x_{ij} \leq |S| - 1, \text{ for every } S \subset V, S \neq \emptyset \tag{3}$$

$$x_{ij} \in \{0, 1\}, \text{ for every } i, j \tag{4}$$

Constraint 2 ensures that every node is visited once and constraint 3 ensures that there are no subtours present.

The Travelling Salesman Problem is a subproblem of many other combinatorial problems, for example in the Vehicle Routing Problem (VRP), where a fleet of vehicles has to service a set of customers from a depot.

Additionally, the TSP has been extended into numerous variations, such as the clustered Travelling Salesman Problem, where every cluster of cities has to be visited once (Lokin, 1979). The Multiple Travelling Salesman Problem, where one can use multiple travelling salesmen to visit all the cities (Berenguer, 1979). One can add time windows, where every city must be visited within a certain time (Lenstra et al., 1988). Of course there is the Asymetric Travelling Salesman Problem, where the distances between cities depend on the direction in which the connection is traversed (Laporte et al., 1987). Its most logical implementation is in the field of logistics, where the Travelling Salesman Problem is the core of every vehicle routing problem. But it has also been used in the fields of DNA Genome sequencing and in drilling and lasering problems (Applegate et al., 2006), and even in creative problems such as a one-line drawing (Kato and Yasuhara, 2000).

Furthermore, the Travelling Salesman Problem has led to a lot of improvements in the field of Mixed Integer Programs (MIP), which are linear programming problems, with the additional constraints that some of the decision

Table 1: Milestones before the TSPlib.

| | | |
|---|---|---|
| 1954 | Dantzig et al. | 49 Cities |
| 1971 | Held and Karp | 57 Cities |
| 1971 | Held and Karp | 64 Cities |
| 1975 | Camerini et al. | 67 Cities |
| 1976 | Miliotis | 80 Cities |
| 1977 | Grötschel | 120 cities |
| 1980 | Crowder and Padberg | 318 cities |
| 1987 | Padberg and Rinaldi | 532 cities |
| 1987 | Grötschel and Holland | 666 cities |
| 1991 | Padberg and Rinaldi | 1002 cities |
| 1991 | Padberg and Rinaldi | 2392 cities |

Source: Applegate et al. (2006)

Table 2: Concorde progress on large TSP instances.

| | | |
|---|---|---|
| 1992 | Concorde | 3083 cities |
| 1993 | Concorde | 4461 Cities |
| 1994 | Concorde | 7397 Cities |
| 1998 | Concorde | 13509 Cities |
| 2001 | Concorde | 15112 Cities |
| 2004 | Concorde | 24978 cities |
| 2004 | Concorde | 33810 cities |
| 2006 | Concorde | 85900 cities |

Source: Applegate et al. (2006)

variables can only be integers. Many of the MIP solvers today were first applied and studied on the TSP (Applegate et al., 2006). Particularly the branch and bound method has its origin with the Travelling Salesman Problem. The idea of repeatedly splitting the problem into smaller sets came from the TSP. When one, for example, has to find the shortest tour between all cities in the United Kingdom, and we do not know whether the link between Edinburgh and Livingston should be part of the optimal tour, one could then separate the solution space into all tours that use this edge and all tours that do not. Before then splitting the problem further, a bound is first computed, such as the sum of the cost of all edges that we said are part of this branch. Using these bounds, certain branches can be excluded and a solution can be found more quickly.

The Travelling Salesman Problem is also important as it is part of the group of NP-complete problems (provided $P \neq NP$). Many important and difficult problems belong to this same set and often solution approaches for these problems share resemblances. Also, all problems in NP are reducible to the TSP. Developing good solution approaches to the Travelling Salesman Problem may in time offer inspiration for solution approaches for other difficult problems; a thing that has often happened in the past.

Most of the earlier instances were inspired by real-life geometric examples, such as Ulysses's trip or travelling by all the capitals of the United States. Chess knight tours were also popular for generating TSP instances. Later randomly generated instances in a euclidean plane also gained popularity and other variations, such as the use of pseudo-distances instead of geometric distances.

One of the earlier larger problems was an instance of 49 cities in the United States by Dantzig et al. (1954). From that point until the 1990s larger and larger problems were investigated. In this time research into the TSP flourished and larger instances were solved. Some of these milestones can be seen in table 1. All of the benchmark instances have since been gathered by Reinelt (1991) and are now part of the TSPlib. When it was published at first, it consisted of only 24 problems. Nowadays it consists of 110 Symmetric TSP instances. In this paper we will look at several (smaller) instances part of this library. A description of these problems can be found in table 9 the appendix.

After the introduction of the CONCORDE TSP solver by Applegate et al. (1998), many large instances have been solved. Some of those can be found in table 2.

Currently, the world TSP tour, consisting of 1,904,711 cities is being solved. This problem was constructed by the University of Waterloo in 2001. An initial solution was found by Karl Helsgaun, which turned out to be 0.076% larger than a lower bound established by the CONCORDE solver in 2007. Currently, the record is held by Keld Helsgaun with a tour length of 7,515,772,107, found on March 13, 2018.[2] This solution is 0.0474% above the best found lower bound. The problem still remains unsolved to this date.

One might wonder why this need to solve these larger problems still remains, as no one will ever actually need the shortest tour around all cities in the world. However, instances of this size are present in the wiring of Printed Circuit Boards in Very Large Scale Integration (VLSI), where the amount of points that need to be connected can grow to sizes this large.

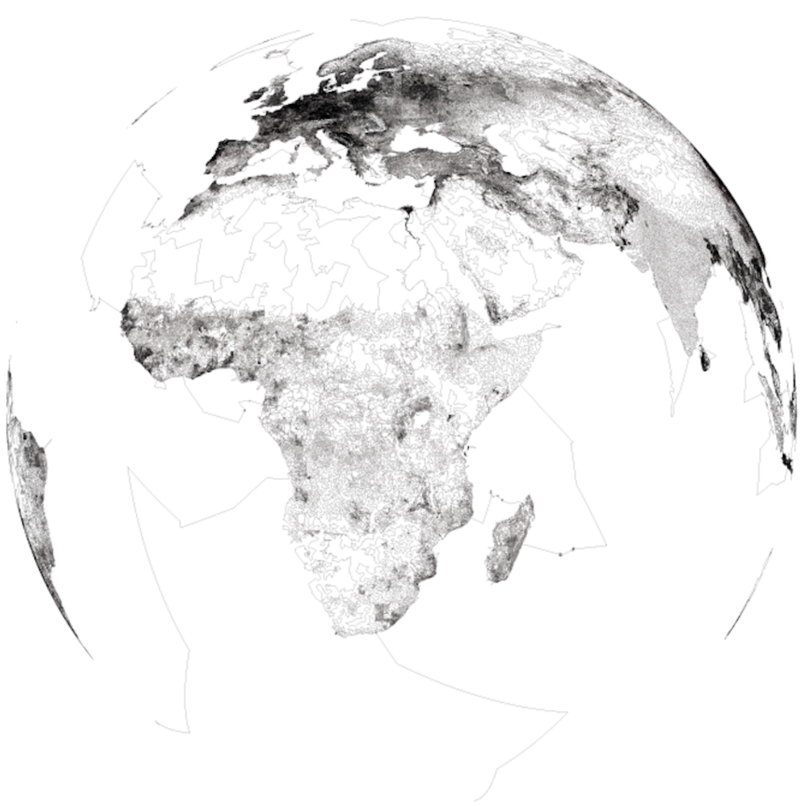
In the next subsections we will discuss several exact and heuristic approaches to solve the Travelling Salesman Problem.

## 2.2 Exact Solution Methods

Exact solution methods are methods which are certain to find the optimal solution within a bounded number of steps. For example for the TSP we know the amount of steps is bounded by simply checking all the possible tours, but as we have seen already, enumerating everything takes a long time. Many exact algorithms deal with relaxing some of the constraints to find lower bounds on the solution and then trying to make the solution feasible again.

The first MIP formulation was constructed by Dantzig et al. (1954) (DFJ), which is the same as the formulation used in section 2.1. This formulation was later strengthened by Miller et al. (1960), by improving the subtour elimination constraints by using additional variables. Later, it was improved even more by Desrochers and

Figure 1: The LKH 2003 world tour.



[2]http://www.math.uwaterloo.ca/tsp/world/

Laporte (1991).

Dantzig et al. (1954) solved their 49 city problem by using the LP formulation, where the decision variable $x_i$ doesn't need to be binary, but $1 \geq x_i \geq 0$. This leads to a lower bound of the optimal value. Later they developed an initial cutting plane or branch and cut method, which became the dominant solver for the Travelling Salesman Problem and other (M)IP problems. The two decades following DFJ however, branch and bound methods got more popular. Why the branch and cut method was not investigated more at that time, is unknown (Applegate et al., 2006).

The most notable branch and bound method was developed by Held and Karp (1970). Held and Karp use a 1-tree to construct a good lower bound on the Travelling Salesman Problem. A 1-tree is a minimal spanning tree with an additional node that connects to the tree with two edges. The authors observe that a TSP tour is actually a one tree with the extra property that every node has a degree of two. This means that the cost of a 1-tree provides a lower bound on the objective value of the TSP. Additionally, they observe that adding a constant to all intercity distances does not change the TSP solution, but may change the 1-tree solution. This provides a large set of lower bounds and, as 1-trees are not costly to produce, a good lower bound can easily be found.

Earlier Held and Karp (1962) used a dynamic programming approach to solve the Travelling Salesman Problem. Using the recursive property that, given a certain part of the optimal tour, the remaining edges must also be chosen optimally. Using this Held and Karp showed it is possible to solve any TSP instance with a worst case running time of $n^2 2^n$. This is still slow, but it is already performing better than the $n!$ running time that follows from enumerating all solutions.

The cutting plane method by Dantzig et al. (1954) was first revisited by Hong (1973) and later by Crowder and Padberg (1980) and Grötschel and Holland (1987). The main idea of the cutting plane method is to start with a linear programming relaxation and then 'cut' the polytope defining the feasible region to make the relaxation stronger. Later Padberg and Rinaldi finalised the branch and cut method in their 1991 paper. The branch and cut method is also the main method in the CONCORDE solver.

## 2.3 Heuristic Solution Methods

Heuristic approaches aim to find a good solution within reasonable time, instead of finding the optimal solution with certainty, but in a longer time. The Lin-Kernighan algorithm is such an algorithm. The LK algorithm belongs to the class of tour improvement algorithms, where an initial tour is taken and one tries to improve this tour to get a good one. Besides this there are tour construction algorithms, which try to construct a tour from the bottom up, and composite algorithms, which combine both of these features.

### 2.3.1 Construction Heuristics

One of the earliest and intuitively most simple algorithms is the nearest neighbour algortihm. Here, one starts at a certain city and travels from city to city consecutively, taking the shortest edge to a city not visited before. When all cities have been visited the tour is closed up to the starting city again. This heuristic has a large drawback that in the beginning the edges chosen will be short, but not towards the end. One tends to paint oneself into a corner where in the end only long edges will remain (Applegate et al., 2006). This algorithm generally doesn't perform well.

A heuristic similar to the nearest neighbour algorithm is the greedy algorithm. Instead of searching globally, we simply start with no edges selected (and thus sort of with $n$ paths of length zero) and keep looking for the shortest edge that links two paths together. Tours are not allowed to be formed before the final step.

Another form of construction heuristics are insertion heuristics. Here the idea is to start with a starting subtour of length $k < n$ and then inserting the remaining nodes into the tour to arrive a good tour. There are many different rules for choosing the next node to be inserted and the difference in performance between these problems is very dependent on the problem it is applied to (Reinelt, 1994).

Another famous construction algorithm is the Christofides algorithm by Christofides (1976). It consists of finding a minimum spanning tree in the graph. Then it finds a perfect matching between the nodes with an odd degree in this minimum spanning tree. When these edges are combined, a Eulerian cycle can be formed (a cycle in which every edge is visited only once, but node may be visited multiple times). And then from this Eulerian cycle an Hamiltonian cycle (and thus a valid TSP tour) can be created by not visiting the repeated nodes twice. The Christofides algorithm runs in $\mathscr{O}(n^3)$. Additionally, as the perfect matching is at most half of the length of the optimal tour and the mininal spanning tree is smaller than the optimal tour one can show that the length of the Christofides tour is at most 50% above the length of the optimal tour.

The final construction heuristic is the savings heuristic by Clarke and Wright (1964). It was originally developed for a vehicle routing problem, but it can also be used for the Travelling Salesman Problem. It mainly consists of choosing a base node and constructing the $n-1$ paths to all the other nodes using one edge. Afterwards paths are merged until there is one large tour. The decision on the next tours to be merged is based on which one gives the largest savings.

Reinelt (1994) performed a relative performance analysis on these construction algorithms and finds the savings algorithm to deliver the best relative performance. No other algorithm was able to get results closer than 2% to the saving heuristic's result. In absolute terms, Reinelt found the savings heuristic to be around 11 - 12% above the optimal values.

### 2.3.2 Improvement Heuristics

Improvement heuristic take a starting tour and try to improve the tour. Simple heuristics are the node and edge insertion algorithms. Here a node (edge) is removed from the tour and all possible location to insert the node (edge) back into the tour are investigated and the one with the highest savings is chosen. Checking all options takes $\mathscr{O}(n^2)$, but the running time of the complete algorithms may be a lot longer. In the worst case one would only find consecutive small improvements and the running time would then depend on the length of the tour and not on the problem size, making it impossible to bind the worst case running time (Reinelt, 1994).

Other improvement algorithms are the 2-opt (Croes, 1958), 3-opt (Lin, 1965), ..., k-opt exchange improvement algorithms. Where $k$ edges are deleted from the tour and new edges are searched for in order to link up the tour again and find a smaller tour length.

Lin and Kernighan (1973) generalised and revolutionised this approach, by iteratively looking for a $k$, the number of edges to swap, resulting in a variable $k$-opt algorithm. The complete algorithm and further literature on the algorithm will be discussed in section 3.

A way of improving the tour before starting an improvement method is by crossing elimination. As when two edges cross in a tour in a euclidian case, there is always an improvement to be gained. Normally checking all improvement options would take $\mathcal{O}(n^2)$, but Ottmann and Widmayer (1990) found a way to do this in $\mathcal{O}(n \log n)$.

## 2.4 Other Heuristics and Genetic Algorithms

The problem with heuristic approaches is that the heuristic may find the optimal solution, but the chance that they instead end up in a local optimum is rather large.

A way to prevent this is to include randomness in the algorithm, such that local optima may be escaped. This is for example done in the iterated Lin-Kernighan algorithm, where a random 4-opt move is performed when a local optimum is found. This makes it more likely that it the end the global optimum is found. Often these algorithms are meta-heuristics that are also applicable to other problems. These algorithms may also draw inspiration from nature, such as genetic algorithms.

An example of such an algorithm for the Travelling Salesman Problem is the Simulated Annealing algorithm, which draws its inspiration from physics. In physics annealing is the process of slowly reducing the temperature until from first being able to reach all states, the material reaches a steady state. It was developed by Kirkpatrick et al. (1983) and follows a similar idea to real life annealing. The algorithm has a temperature parameter that slowly cools down with time. The algorithm starts with a solution $x$ and tries to find solution in the neighbourhood of $x$. If $y$, the found solution, is better than $x$, $x$ becomes $y$. However, if $y$ is worse, $x$ can still become $y$, but the probability that this happens depends on the temperature. When the temperature is still high, these suboptimal moves will be frequent, but as the temperature falls, these moves will be less frequent. Simulated Annealing has been tried on the Travelling Salesman Problem by a couple of researchers, among which Bonomi and Lutton (1984) and Golden and Skiscim (1986), but performances have been inconsistent and successes not clear (Reinelt, 1994).

Glover's (1989, 1990) Tabu Search is quite similar to Simulated Annealing, as the objective function is sometimes allowed to increase as well. Additionally, a Tabu list is kept with previously found solution such that these are not being visited again. Its successes for the Travelling Salesman Problem depend on a careful and specific setting of the Tabu parameters, but it can be a very well-performing algorithm (Reinelt, 1994).

Another popular group of algorithms are genetic algorithms developed by Holland (1992) and Goldberg and

Holland (1988). Genetic algorithms are meta-heuristics that take inspiration from natural selection. The algorithm starts with a population of solutions, each with an own fitness score, which in most cases is the objective value. Then for a fixed number of times or when a certain threshold is met, new populations are made. Solutions have the possibility to 'procreate' or 'crossover', where two solutions are combined into new solutions, or to 'mutate', where one solution is perturbed on itself. Mostly, solutions with a better fit score are more likely to procreate than others. The crossover part of the algorithm attempts to improve solutions while the mutation part is meant to escape local optima. The main challenge for designing genetic algorithms for the Travelling Salesman Problem is the way to combine tours together and how to mutate tours. For more information on this, see Potvin (1996) and Larranaga et al. (1999).

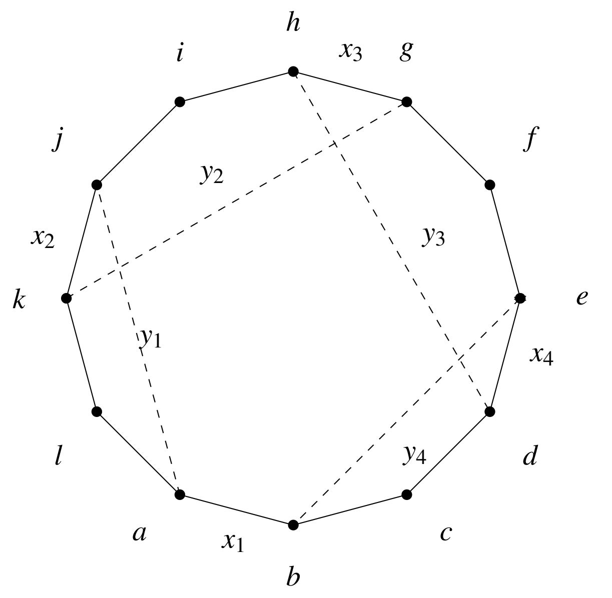
Dorigo and Gambardella (1997) took inspiration from nature as well, particularly ants, in inventing their Ant Colony optimisation algorithm. In real life, ants release pheromones when they find a food source, which leads to other ants following the same trail to the food. The more pheromones there are on a trail, the more ants will come release pheromones again, reinforcing the effect. In the algorithm, a colony of ants starts at different cities and goes on to make a tour in a nearest neighbour fashion. However, instead of always choosing the shortest edge, the choice for an edge depends on the length of the edge and the amount of pheromones on the edge. Every time an ant travels an edge, it releases pheromones, and after the colony has completed the tours, the ant with the smallest total tour length releases additional pheromones on the tour's edges. This method has since been improved by Escario et al. (2015), and Mahi et al. (2015) by using the particle swarm optimisation by Wang et al. (2003) and 3-opt moves.

Other more recent applications of genetic algorithms on the Travelling Salesman Problems are a FireFly algorithm implementation by Jati et al. (2011), a Cuckoo Search by Ouaarab et al. (2014), a bat-inspired algorithm by Saji and Riffi (2016) and an American Buffalo optimisation implementation by Odili and Mohmad Kahar (2016).

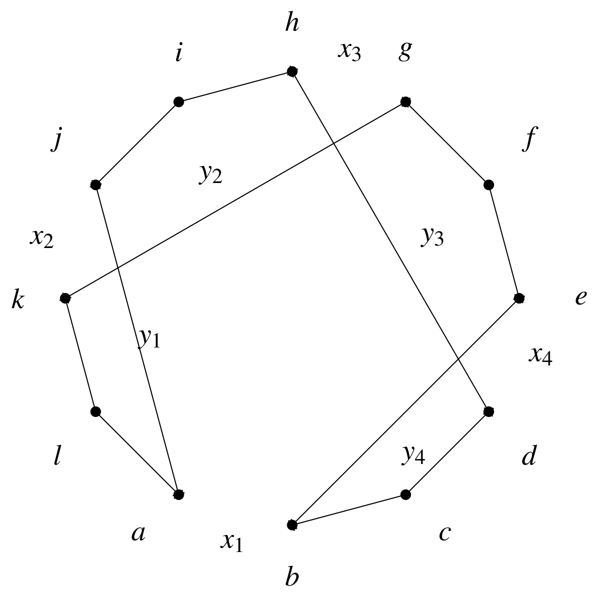
## 3 Lin-Kernighan Algorithm

Lin and Kernighan (1973) generalised and revolutionised the $k$-opt approach, by dynamically looking for a $k$, the number of edges to swap. Thus, the main principle behind the LK algorithm is that instead of always making a fixed amount of changes, it sequentially tries to find the edges to swap by starting with the edges that are most out of place and continue until the proposed swap is longer profitable. Let $S$ be the set of all $\frac{n(n-1)}{2}$ edges between the $n$ cities, and $T$ an arbitrary tour that satisfies the tour condition. The aim is to find two sets of edges $X = \{x_1, x_2, ..., x_k\}, x_i \in T$ and $Y = \{y_1, y_2, ..., y_k\}, y_i \in S/T$, so that removing the edges in X and adding the edges in Y results in a tour $T'$ with a lower total cost. ($F(T') < F(T)$). Such a swap for $k = 4$ is given in figures 2.

Figure 2: Sequential 4-opt swap



: The tour before the sequential 4-opt move: a-b-c-d-e-f-g-h-i-j-k-l-a



: The tour after the sequential 4-opt move: a-j-i-h-d-c-b-e-f-g-k-l-a

Define the gain by swapping $x_i$ with $y_i$ as $g_i = |x_i| - |y_i|$, where $|x_i|$ and $|y_i|$ are the lengths of $x_i$ with $y_i$. Then the total gain of making a $k$-opt swap is $G_k = \sum_1^k g_i = F(T) - F(T') > 0$. Even though some individual swaps may be negative ($g_i < 0$), the total sum can be positive. This means we can continue searching as long as the sum of gains is positive and prevents the algorithm from stopping too early and missing any fruitful swaps and ending up in a local minimum too soon.

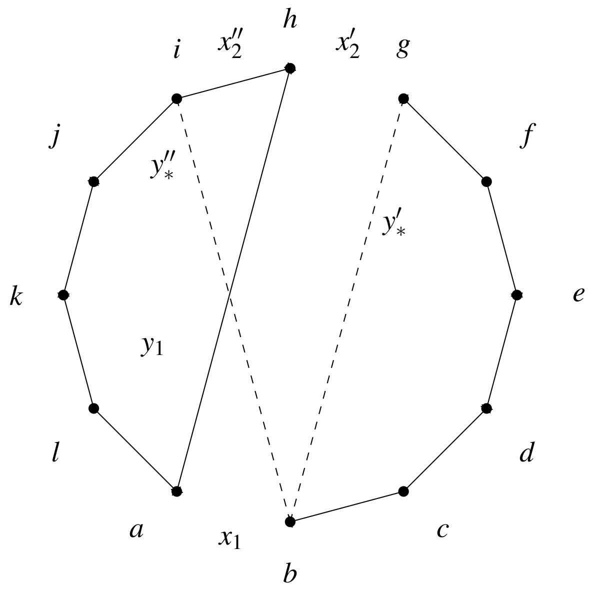
In order to make the algorithm work, Lin and Kernighan observe several things that need to be satisfied:

- If we are at a certain $k$ in the algorithm and want to stop searching, we need to know that the exchange results in a feasible tour. In order to do this easily and reduce programming bookkeeping, we require that every newly added swap can link back up to the starting node to form a feasible tour.

- the sets X and Y are disjoint, which means that any edges that were previously assigned to be removed cannot be assigned to be added and vice versa.
- The gain function $G_i = \sum_1^i g_i$ represents the gain of the swap up until the point $i$. We require a positive total gain for each new swap $y_i$. When at some point $G_i$ is negative we have reached the stopping rule and we can stop searching.

The first bullet point is an important feature of the Lin-Kernighan algorithm, i.e. all swaps must sequential. This means that for every $y_i$, the next $x_i$ is uniquely determined, as there can be only one $x_i$ after which the tour can be closed up again. This can be seen in figure 3. Starting in node b, edge $x_1$ = (b, a) is deleted, after which edge $y_1$ = (a,h) is added. Now we see that there can be only one choice for $x_2$. When we would choose $x_2'$, the resulting $y_*'$ results in two subtours. Therefore, the only choice for $x_2$ is $x_2''$, because then the tour can be closed up again by having $y_2 = y_*''$ = (i, b). This feature means that a non-sequential swap as seen in figure 4 can never take place. This means that some good swaps might be ruled out. However, Lin and Kernighan found non-sequential swaps to be very infrequent in problems they analysed.

Figure 3: Uniquely determining $x_i$

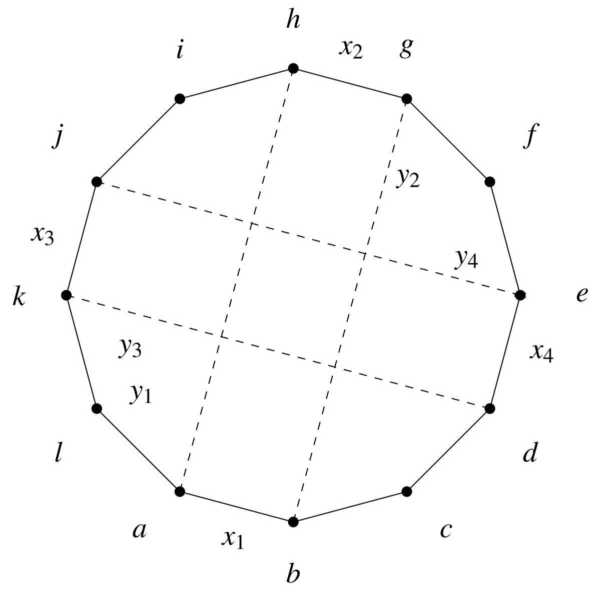


The complete Lin-Kernighan algorithm can be seen in figure 5.

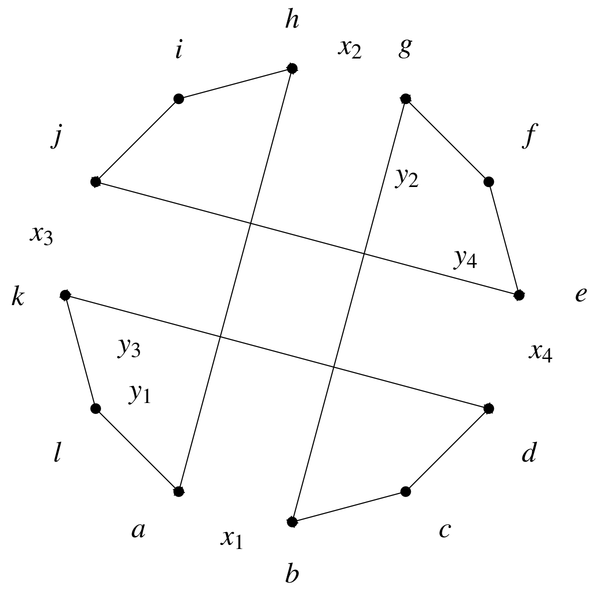
In step 9, the candidate $y_i$'s are restricted to the five nearest neighbours of $t_{2i}$, as, according to Lin and Kernighan, one could expect these links to be most likely part of the optimal tour. This saves investigating all the possible $y_i$'s, saving computation time. On the other hand it goes at the cost of possibly not finding optimal links.

Lin and Kernighan usually find a small number of local optima, ranging from 2 to 5. This suggests that the edges that are a part of all those solutions are likely to be part of the optimal solution. Therefore, in step 8, the

Figure 4: Non-Sequential 4-opt swap



: The tour before the non-sequential 4-opt move: a-b-c-d-e-f-g-h-i-j-k-l-a



: The tour after the non- sequential 4-opt move: a-h-i-j-e-f-g-b-c-d-k-l-a

deletion of edges is restricted to edges that are not part of those previously found best tours. In order to prevent too much bias towards these solutions, this rule is only applied when $i$ is larger than 4.

In step 12, the search is stopped when the tour is similar to a previously found local optimal tour $T$. This is to prevent time wasted on continuing to try to improve this tour, which has already been done before: this is the so-called *checkout time*. Lin and Kernighan state that computation times are reduced with so much as 30 to 50 percent due to this rule.

In step 9, we see that not only the smallest $y_i$ is chosen, but the algorithm also investigates the length of $x_{i+1}$, the node to be deleted after. This prevents the case where small $y_i$ is added, but thereafter removing an edge $x_{i+1}$ that was actually a good edge in the tour. Additionally, Lin and Kernighan state that the algorithm becomes more global of nature, as merely looking at the length of $y_i$ implies a nearest neighbour approach, while also looking at the next $x_{i+1}$ entails information about the current tour as well.

Of course for $i = 1,2$ the situation of multiple possible $x_i$'s (only for i = 1) and $y_i$'s arises more frequently. A backtracking procedure is applied in step 10. Backtracking is a trade-off between computation time and solution quality, as complete backtracking would find the optimal solution. Lin and Kernighan experimentally found backtracking up to a depth of 2 to be optimal. Additionally, only five candidates are considered for $y_1$ and $y_2$, as this halved computation time and barely affected performance.

In the original implementation also a 4-opt non-sequential move is investigated as a 'cheap insurance' in the case such an optimal swap would be present. In our implementation, this step is skipped. In this paper however, we will only implement the algorithm as described in this section.

Figure 5: The Lin-Kernighan heuristic

```
1. Create an initial random starting solution $T$.
2. Set $G^* = 0$, the best improvement found so far.
3. Set $i = 1$
4. Choose $t_1$, the starting node in the tour.
5. Choose edge $x_1 = \{t_1, t_2\}$, where it is the largest of the two possible edges.
6. Choose edge $y_1 = \{t_2, t_3\}$, such that $G_1 > 0$. If this is not possible, go to step 13.
7. Set $i = i + 1$
8. Choose $x_i = \{t_{2i-1}, t_{2i}\}$, the next edge to be removed, such that the following conditions are
   satisfied:
    (a) The tour can be closed by choosing $y_i^* = \{t_{2i}, t_1\}$, such that the tour closes up again.
    (b) the edge that is removed is not an edge that has previously been added.
    (c) If $i > 4$, $x_i$ cannot be part of the intersection of the sets of edges of the previously
        found best solutions.
   Check whether the gain by closing up the tour is larger than the best gain found so far
   ($G^*$), if so, set $G^* = G_{i-1} + |x_i| - |y_i^*|$.
9. Choose $y_i = \{t_{2i}, t_{2i+1}\}$, the edge to be added, such that the following condition are
   satisfied:
    (a) $G_i = \sum g_i > 0$, the total gain by adding the new edge is larger than zero.
    (b) the edge that is added is not an edge that previously has been removed.
    (c) $x_{i+1}$ exists; there is a next edge which can be removed.
    (d) $t_{2i+1}$ is part of the set of five nearest neighbours of $t_{2i}$.
   For the feasible $y_i's$, choose the five $y_i$'s for which $|y_i|$ is smallest.
   For these five $y_i$'s, choose the $y_i$ for which $|x_{i+1}| - |y_i|$ is maximum.
   If a $y_i$ has been found, go back to step 7.
10. If $G^* = 0$ (No gain has yet been found) do:
    (a) If there is an untried edge $y_2$ and fewer than 5 options for $y_2$ have been
        investigated, set $i = 2$ and go to step 9.
    (b) If there is an untried edge $x_2$, set $i = 2$ and go to step 8.
    (c) If there is an untried edge $y_1$ and fewer than 5 options for $y_1$ have been
        investigated, set $i = 1$ and go to step 6.
    (d) If there is an untried edge $x_1$, set $i = 1$ and go to step 5.
11. If $G_i < G*$ or no more feasible $x_i$ and $y_i$ can be found. Make the swap corresponding to $G*$
    resulting in tour $T'$.
12. If tour $T'$ is the same as a previously found best tour, the search is stopped.
13. Continuing with tour $T'$, if there are still untried alternatives for $t_1$, go to step 2.
14. Store the final tour $T'$ and corresponding objective value $F(T')$ and either stop or go to
    step 1 and investigate more tours.
```

## 3.1 Further Literature on the Lin-Kernighan Algorithm

In this section we will discuss all the revisions and improvements the Lin-Kernighan algorithm has seen so far. Since its initial publication in 1973, the Lin-Kernighan algorithm has been implemented and altered by many researchers. Many implementations have sought to simplify the implementation of the LK heuristic, as the implementation of the original algorithm can be rather tedious. Mak and Morton (1993) seek to improve the algorithm by allowing temporary infeasibility in the sequencing and with that allowing for more breadth in the search. In 1990, Johnson successfully implemented the algorithm, with the alteration of allowing the addition of previously deleted $x_i$'s in step 9 of the algorithm.

Applegate et al. (1990) and Reinelt (1994) have tried to limit the depth of the search of the heuristic to 50 and 15 steps respectively. However, as this depth was already rarely reached in the original implementation, effects were inconsequential.

In 1992, Martin et al. attempted an improvement to the LK heuristic in a genetic fashion. The population consist of only one 'creature' and therefore only mutations are possible. In the attempt to escape local minima, there is small chance or permutating the tour by performing a non-sequential 4-opt move such as seen in figure 4. The authors try to tackle the problem that the amount of local optima grows fast with the problem size and thus the computation time necessary for being certain of having found a global optimum grows as well. This is attempted by temporarily allowing an increase in tour length in order to increase the search space in the popular Simulated Annealing way. Applegate et al. (2003) improved this method even further, resulting in the popular Chained Lin-Kernighan (CLK) algorithm.

Additionally, Applegate et al. (2003) looked into the level of backtracking. Whereas the original algorithm considers 5 options for the first swap and 5 options of the second swap, Applegate et al. consider several variations of depth and number of options for backtracking. They find Lin and Kernighan's rule to work rather well.

For a more extensive review of the alterations of the Lin-Kernighan heuristic done before 1997, see Johnson and McGeoch (1995).

The most widely used implementation of the Lin-Kernighan heuristic is the Lin-Kernighan-Helsgaun (LKH) implementation by Helsgaun in 2000, which is also different in design from the original algorithm. Helsgaun finds some defects in Lin and Kernighan (1973)'s refinement heuristic rules to direct the search:

- For finding a new $y_i$, Lin and Kernighan restrict the options to the five nearest neighbours. Even though this intuitively might seem like a good rule, as these shorter edges are more likely to be part of the optimal tour, Helsgaun finds that often optimal tours consist of a nearest neighbour link of a higher order. This would advocate an increase of the candidate set of edges, but this would also mean a considerable increase in running time. Helsgaun proposes a dynamic measure of 'nearness' (called $\alpha$-nearness) as an estimate of the edge's probability of being in the optimal solution.

- Additionally, Helsgaun already put the restriction of not being part of a found local optimum on $x_1$, in addition to for a depth larger than 4.
- The basic move in the algorithm now is a 5-opt move and the search of a move is directly stopped when the close up of the tour results in improvement. Experiments showed that this made backtracking unnecessary (except for $x_1$). Applegate et al. (2006) cites this as the most detrimental improvement with regards to the original algorithm.
- To allow for non-sequential 4-opt moves during the move construction, temporary infeasibility is allowed, contrary to only looking for a non-sequential 4-opt move after a local optimum has been found.
- As opposed to constructing random starting tours, Helsgaun constructs candidate tours from the previously found local optima.

After that Helsgaun has improved the LKH for many variations of the Traveling Salesman Problem. In 2009, Helsgaun introduced several improvements to the LKH by

- generalising the idea of using 5-opt moves to allow for non-sequential exchanges to a generalised $k$-opt submoves,
- partitioning large problems into several smaller subproblems,
- attempting to construct the optimal tour by merging some of the found local optima.

These improvements resulted in the LKH-2 variation.

In 2011, Helsgaun extended the LKH-2 to the Bottleneck TSP, where the largest edge in the tour is minimised, in 2014 to the Equality Generalized Travelling Salesman Problem, where the cities are divided up in clusters and every cluster must by visited at least once. Hains et al. (2012) improved the LKH-2 on heavily clustered instances of the TSP, where the LKH-2 was performing badly. Currently, Helsgaun (2000) has improved the LKH to the LKH-3, which solves a huge variety of Travelling Saleman Problems, such as the Sequential Ordering Problem (SOP), the Traveling Repairman Problem (TRP), variations of the Multiple Traveling Salesman Problem (mTSP), and variations of Vehicle Routing Problems (VRPs). Helsgaun bases the LKH-3 on the simple idea of transforming these problems to the standard TSP and imposing a penalty function on the additional constraints. Most recently, Taillard and Helsgaun (2018; 2019) introduced a method to generate candidate solutions for very large problem instances faster.

# 4 The Travelling Thief Problem

Some academics deem modern day operation research problems not to capture common characteristics of real life problems well enough. By not investigating these characteristics, modern day heuristics will not be well applicable to real life problems.

Therefore, Bonyadi et al. (2013) tried to capture one of those characteristics, the interdependence of subproblems, by introducing a new problem: The Travelling Thief Problem (TTP), which is a combination of the Travelling Salesman Problem and the Knapsack Problem. A thief visits a set of cities (all must be visited, as in the TSP), and at every city certain items are located. The thief carries a knapsack, in which he can take the items. All items have a certain profit value and a certain weight. The thief pays rent for the knapsack per time unit. The aim is to maximise the profit of the thief. Picking up more items increases his profit, but in turn also slows him down and increases his renting cost. As such, there is an interdependecy between the two problems, and an optimal solution for either of the subproblems does not gaurantee an optimum for the problem as a whole.

The kind of interdependecy of the Travelling Thief Problem is also present in real life problems. Earlier we talked about the wiring of VLSI computer boards. Sometimes there are certain sensitive parts the wire cannot cross. Clearly there is an interdependency here, as the optimal route might need certain parts to move, possibly making the route suboptimal again.

Bonyadi et al. (2013) initially presented two possible ideas for the Travelling Thief Problem, the TTP$_1$ and the TTP$_2$. The TTP$_2$ consists of two objective functions; minimise total travel time and maximise total knapsack value, where the values of the items drop over time. The TTP$_2$ received little attention in the literature. The most used and studied formulation of the TTP is the one by Polyakovskiy et al. (2014), which is very similar to the TTP$_1$ formulation. The problem is defined and explained in the next section.

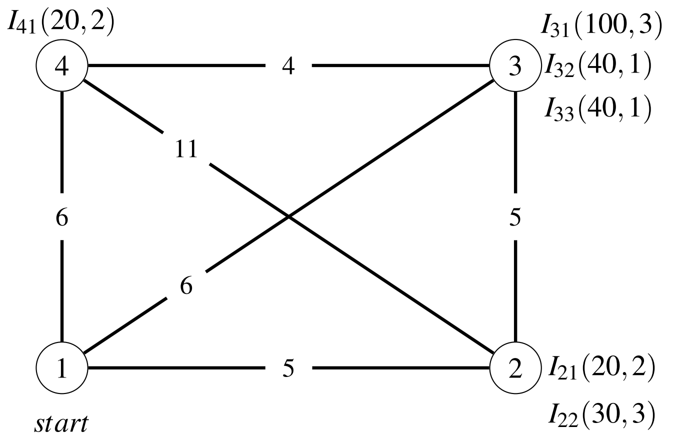
## 4.1 Problem Description

Given a set of $n$ cities $N = \{1,...,n\}$ with distances $d_{ij}$. Every city $i$ except for city 1 contains a set of items $M_i = \{1,...,m_i\}$, and each item $k$ present in city $i$ has a certain value $p_{ik}$ and weight $w_{ik}$. The thief has to travel each city and has the option to pick up items and add these to his knapsack. The knapsack has a total capacity of $W$ and the thief pays a renting rate $R$ per time unit. The maximum speed with which the thief can travel (when the knapsack is empty) is denoted by $v_{max}$ and the minimum speed (when the knapsack is full) by $v_{min}$. The objective of the problem is to find the tour and item picking plan with the highest profit. Decision variables $x_{ij} = 1$ if edge $(i, j)$ is selected in the tour and 0 otherwise and $y_{ik} = 1$ if item $i$ is picked up in city $k$ and 0 otherwise, resulting in a tour $T = \{c_1,...,c_n\}$, $c \in N$ and packing plan $P = \{y_{21},...,y_{nm_i}\}$. Additionally, we set the variable $W_i$ as the weight of the knapsack when the thief departs city $i$. The objective function is then given as

$$Z(T,P) = \sum_{i=i}^{n}\sum_{l=1}^{m_i} p_{ik}y_{ik} - R\Big(\frac{d_{x_n x_1}}{v_{max} - vW_{x_n}} + \sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{v_{max} - vW_{x_i}}\Big) \tag{5}$$

The left part of the objective function entails the profit that is gained from picking up the items. The right part of the expression is the renting rate R times the distance travelled adjusted for the speed the thief is travelling at. Here, $v = \frac{v_{max} - v_{min}}{W}$. This ensures that when the knapsack is empty the thief travels at speed $v_{max}$ and when it is full he travels at speed $v_{min}$.

Figure 6: A TTP instance example



We explain the problem with a small example, seen in figure 6.

We assume the capacity of the knapsack $W$ to be 3, the renting rate R to be 1, the maximum speed $v_{max}$ to be 1 and the minimum speed $v_{min}$ to be 0.1. All nodes except the start city 1 have items. For example, city 2 has two items, item $I_{21}$ with profit $p_{21}$ = 20 and weight $w_{21}$ of 2 and item $I_{22}$ with profit $p_{22} = 30$ and weight $w_{22} = 3$.

Clearly, we can see that solving the two problems separately does not at give us the best solution, as we would choose tour $T' = (1,2,3,4)$ and $P = (0,0,1,0,0,0)$. This would mean: travelling from city 1 to city 2 with cost = 5, from city 2 to city 3 with cost = 5, from city 3 to city 4 with cost = $\frac{4}{0.1} = 40$, and from city 4 back to city 1 with cost = $\frac{6}{0.1} = 60$, resulting a loss of $Z(T,P) = 100 - 5 - 5 - 40 - 60 = -10$. It is important to note that the objective value is not as easily interpreted as in the TSP or the knapsack, as it is artificially made according to the renting rate. Objective values can therefore also turn negative.

The optimal value for this instance is $Z(T,P) = 45$, with $T = (1,2,4,3)$ and $P = (0,0,0,1,1,0)$. The thief travels from city 1 to city 3 via city 2 and 4. In cities 2 and 4 no items are picked up and the thief therefore travels with speed 1 and the cost of this part of the tour 5 + 11 + 4 = 20 (the renting rate is 1). At city 3 the items $I_{32}$ and $I_{33}$ are picked up, resulting in a total profit of 80. The distance from 3 back to the start city 1 is 6, however, as we picked up items, the weight of the thief's knapsack is now 2, slowing the thief down. The thief travels with speed $(1 - \frac{2}{3} \cdot (1 - 0.1)) = 0.4$, thus making the renting cost of the last edge $\frac{6}{0.4 = 15}$. This results in a total profit of 80 - 20 - 15 = 45. However, there is another optimal solution, with $T = (1,2,3,4)$ and $P = (0,0,0,1,1,0)$, also resulting in an objective value of 45.

## 4.2 Benchmark Set

Polyakovskiy et al. (2014) use Reinelt's TSP*lib* to create new benchmark problems. For constructing the items and their properties they use three principles: *uncorrelated, uncorrelated with similar weights* and *bounded strongly correlated types*.

For the *uncorrelated* case, the weights and items are uncorrelated and $w_{ik}$ and $p_{ik}$ are both uniformly distributed integers on the interval [1, $10^3$]. For the *uncorrelated with similar weights* case, the weights for all the items are very near to each other, making the knapsack problem more difficult to solve. Here $w_{ik}$ and $p_{ik}$ integer values uniformly distributed within [$10^3$ , $10^3$ + 10] and [1, $10^3$] respectively. This means that the weights are always larger than the profits. In the *bounded strongly correlated* case, the weight is uniformly distributed on [1, $10^3$] and the profit is given by $p_i = w_i + 100$. This is turn means that the profits are always larger than the weights. For the Travelling Thief Problem we are more interested in the interdependency of the two problems, than in the difficulty of the knapsack problem. However, a more difficult knapsack problem might also mean a stronger and more difficult interdependency. For every type of instance, there are 10 instances with a varying knapsack capacity depending on the total weight of all items. For instance $i$ the $capacity = \frac{i}{11} \cdot totalWeight$. The value for the renting ratio is set as $\frac{\text{optimal value KP}}{\text{optimal value TSP}}$, such that both subproblems have an around equal contribution to the problem's objective value.

Polyakovskiy et al. (2014) also differ in the amount of items per city. (Wuijts, 2018) adds the note that most research nowadays considers the cases limited to one item per city. Additionally, when all cities have more items, the frequency of items with good profit-weight ratios become more frequent and the order of the cities will be of less importance. After several competitions, mainly organised by the university of Adelaide, a comprehensive problem set can be found on the website of the University of Adelaide.[3] The total benchmark set consist of 9720 TTP instances, constructed out of 81 Travelling Salesman instances, 3 Knapsack Problem Types, 4 different item-per-city factors, and 10 different knapsack capacities (and renting ratio's). In this paper we will focus on the 30 *eil51* instances with 1 item per city. The optimal values of these problem are not known yet, but we will use the best values found yet by Wuijts (2018) for comparison.

With regards to the difficulty of the problem, Bonyadi et al. (2013) found with the origination of the problem that the optimum for one of the subproblems does not guarantee overall optimality. We have also seen this in the small example we gave in figure 6. Mei et al. (2016) state that the non-linear dependence of the TSP and the Knapsack problems may make it impossible to decompose the problem into subproblems, and must be solved as a whole. The interdependency makes the TSP not symmetric anymore. This is because now the travel time depends on the weight of the knapsack, which is different depending on the direction in which the tour is travelled. This has as a consequence that checking a swap in the Lin-Kernighan now costs $\mathcal{O}(n)$ time instead $\mathcal{O}(1)$. However, as Wuijts (2018) ponders over the complexity of the TTP and wonders whether it is truly the interdependency that makes

[3]https://cs.adelaide.edu.au/ optlog/CEC2014COMP_InstancesNew/

this problem difficult:

> 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.

Another thing that makes developing solution methods difficult is the fact that by making a small change in a solution by adding an item, or making a 2-opt swaps affects the travel time immensely and the effect cannot be traced back in a constant time (El Yafrani and Ahiod, 2017). It is therefore impossible to do an "incremental fitness analysis for the most trivial operators" (Wuijts, 2018). In the next section we will discuss the solution methods that have been developed to this date.

## 4.3 Solution Methods

Wu et al. (2017) have set out some exact solution methods in order to find global optima for the TTP benchmark instances. These are needed to compare the quality of the growing set of heuristic approaches to the TTP. They managed to find some optimal values for small ($n < 20$) instances. A good exact solver is still not available (Wagner et al., 2018), however, several heuristic methods for solving the TTP have been developed and these are discussed below.

### 4.3.1 SH and DH Solvers

Together with the publication of the benchmark set, Polyakovskiy et al. (2014) also proposed a simple construction algorithm for the TTP: the Simple Heuristic (SH). The SH first creates an optimal tour according to the normal intercity distances and then scores all items with

$$s_{x_ik} = p_{x_ik} - Rt_{x_ik},$$

where $t_{x_ik}$ is a speed adjusted measure from the city the item can be picked up until the end of the tour, given by

$$t_{x_ik} = \frac{d_{x_i}}{v_{max} - vW_{x_ik}}.$$

It does not take into account a decrease in speed due to items that are picked up afterwards. Additionally a fitness value

$$u_{x_ik} = Rt' + (P_{x_ik} - Rt_{x_ik}),$$

where $t'$ is the length of the tour and $t'_{x_ik}$ is the length from city $x_i$ to the end of the tour with the speed adjusted for only picking item $I_{x_ik}$. If this value is negative, then it better to not pick any items than to pick this item only. These items are excluded for the item selection based on $s_i$. The RLS and (1+1)-EA attempt to iteratively improve the packing plan given a tour.

Bonyadi et al. (2014) propose an algorithm very similar to the SH heuristic. The DH solver, is the same as the SH heuristic, but then without the fitness evaluation of $u_{x_ik}$.

If we apply the DH algorithm to the example given in figure 6, we find the shortest tour $T = (1,2,3,4)$ (or $(1,4,3,2)$, but here we will focus on one for examplary purpose).

For item $I_{21}$, the profit is 20, and the remaining distance until the end of the tour $d_{x_i}$ = 15. However, due to picking up the item, the weight of the knapsack is increased by 2, and the speed travelled is

$$v_{max} - vW_{x_ik} = 1 - \frac{0.9}{3} \cdot 2 = 0.4. \tag{6}$$

And therefore

$$t_{21} = \frac{15}{0.4} = 37.5. \tag{7}$$

So the score of item $I_{21}$ = 20 - 37.5 = -17.5. Note that the renting rate was set to 1 in the example, so the R is omitted from the formula. If we calculate the scores of all items, we find the following ranking of items:

$$s_{32} = 40 - \tfrac{10}{0.7} = 25.7 \qquad s_{31} = 100 - \tfrac{10}{0.1} = 0$$

$$s_{33} = 40 - \tfrac{10}{0.7} = 25.7 \qquad s_{21} = 20 - \tfrac{15}{0.4} = -17.5$$

$$s_{41} = 20 - \tfrac{6}{0.4} = 5 \qquad s_{22} = 30 - \tfrac{15}{0.1} = -120$$

After this, the knapsack is filled according to the scores. This means items $I_{32}$ and $I_{33}$ are picked up. Resulting in an objective value of $Z = 40 + 40 - 5 - 5 - \frac{4}{0.4} - \frac{6}{0.4}$ = 45, which is the optimal value of the problem.

For the SH algorithm, the $u$-scores would also be calculated, and the renting ratio times the length of the tour would be added to the scores. In this case this would be 20. If for any of these items this score is negative, it is not considered in the packing routine. In this example this would only be the case for item $I_{22}$, with $u_{22}$ = -100.

### 4.3.2 The PACKITERATIVE algorithm

Faulkner et al. (2015) also investigated the method of first picking a tour and then optimising through a packing heuristic with their PACKINGITERATIVE algorithm. It is similar to the DH solver in the way that the items are rated on the profit-weight ratio as well as their distance, but in a different way:

$$score(I_{x_ik}, \alpha) = \frac{p^{\alpha}_{x_ik}}{w^{\alpha}_{x_ik} \cdot d_{x_i}}, \text{ where } d_i \text{ is the total remaining length of the tour.}$$

For every item that is added, the authors compute the objective value and add items to the packing plan until no more gain is achieved. The $\alpha$ is used to control the balance between the profit-weight ratio and the distance, as for different problem instances the one might be more important than the other. The optimal $\alpha$ is sought in the PACKITERATIVE algorithm in an iterative way. After constructing an initial tour and packing plan, they use two

operators to improve a given solution with a local search routine: BITFLIP, a greedy procedure, where for every item changing the binary picking variable an improvement on the objective value is checked. And INSERTION, where it is attempted to insert a city that contains a good items that is being picked at the end of the tour. Here there is a trade off between increasing the tour length and reducing the travel time due to the decrease in weight of the knapsack over a certain part of the tour.
We will showcase how the algorithm works according to the example in figure 6. Choosing $\alpha = 3$, the score for item $I_{21}$ is given by:

$$score(I_{21}, 3) = \frac{20^3}{2^3 \cdot 15} = 66.6.$$

We will not give the score for all items. The items with the highest scores are items $I_{32}$ and $I_{33}$ and it thus finds the same solution as the DH and SH solvers in this case.

### 4.3.3 Other Solution Methods

Aside from the construction algorithms described above, there exist several other solution methods. Together with the publication of the Simple Heuristic (SH), Polyakovskiy et al. (2014) proposed a Random Local Search (RLS) and a (1+1) Evolutionary Algorithm ((1+1)-EA) to solve the TTP.

Bonyadi et al. (2014) propose a second algoirthm along with the DH heuristic: the COSOLVER. The COSOLVER is a socially inspired algorithm, which attempts to solve the two problems independently and then iteratively one-at-a-time using communication between the two programs. The authors show that the COSOLVER in most cases outperforms the DH algorithm. This might also be due to the fact that they analyse small instances and the COSOLVER uses an exact approach to the subproblems while the DH solver does not.

Mei et al. (2016) made several extensions to Bonyadi et al. (2014)'s COSOLVER algorithm. Wagner et al. (2018) attempted to solve the TTP with the swarm intelligence Ant Colony approach. El Yafrani and Ahiod (2018) applied a memetic hill climbing algorithm and a simulated annealing algorithm to the TTP.

In order to compare our own algorithm, which is explained in the next subsection, with other construction algorithms, we implement the DH, SH and Faulkner et al. (2015)'s PACKITERATIVE algorithms.

## 4.4 Lin-Kernighan Modified Distance Algorithm

In this paper we propose a new construction algorithm for the Travelling Thief Problem that tries to use the interdependency of the two problems. It is similar to the other constructors available, in the sense that we optimise one subproblem, and then take properties of the first subproblem into account when solving the other subproblem.

The constructors present now, the SH heuristic, the DH heuristic, and PACKITERATIVE all have in common that first a tour is constructed, and then a packing plan is constructed taking the tour into account.

However, instead of first optimising the tour, and afterwards trying to find a good packing solution, we do the opposite: We first decide which items are being picked, and take this into account when constructing the tour. Our algorithm, the Lin-Kernighan Modified Distance Algorithm (LKMD) is based on a couple of principles:

- Items with a high profit-weight ratio give the highest profit per unit of weight. In turn a higher weight results in a longer travel time, so these items also give the highest profit per time unit, depending on the position of the city in a tour.
- In the case that all intercity distances would be the same, we would prefer the cities containing the items in the packing plan to be visited at the end of the tour.

Of course the second point is not true in TSP or TTP instances. We do see that there is a trade off between the two extremes of: on the hand the naïve approach of picking the best items and the shortest tour distance-wise, and on the other hand picking the best items and visiting these cities last necessarily. In the algorithm we try to find the optimal trade-off between these two situations. We do this by modifying the distance matrix that is used as input to find the TSP tour.

Picking the items first divides our set of cities $N$ up in three subsets, $N_h$, the set containing the home city 1, $N_1$, the set of cities that have their items picked up, and $N_0$, the set of cities whose items aren't picked. In order to have the cities in $N_1$ at the end of the tour, we want to decrease the distances between those cities, as well as the distance to the starting (and thus ending) node, as well as decrease the distance from cities in $N_1$ to cities in $N_0$. These distances are then used as input for the LK algorithm. A description of the algorithm is seen in figure 7.

Firstly, all items are ranked in non-decreasing order based on the profit-weight ration. Then the knapsack is filled with the items with the highest scores. For a certain percentage value C, the distances between cities that have items that are being picked are being reduced. Also, the distances from the home city to these cities are reduced. The distances between cities that have picked items and cities that have no picked items are being increased by this percentage. Then, a tour is found with the LK algorithm using these pseudo-distances. We iteratively try to find the value of C that finds the best objective value, similarly to the PACKITERATIVE.
Items $I_{32}$ and $I_{33}$ have the highest profit-weight ratio's, so these two items are selected. If we set C = 75%, the distances are mutated as follows:

$$d'_{12} = d_{12} = 5 \qquad d'_{13} = (1-C)d_{13} = 1.5$$

$$d'_{14} = d_{14} = 6 \qquad d'_{23} = (1+C)d_{23} = 8.75$$

$$d'_{24} = d_{24} = 11 \qquad d'_{34} = (1+C)d_{34} = 7$$

And the optimal tour found is $T = (1,2,4,3)$, which corresponds to one of the optimal solution described in the example case.

This showcases that even though the LKMD algorithm works in a different way and finds different solutions, it might also be able to find good solutions to the TTP.

Figure 7: The Lin-Kernighan Modified Distance algorithm

```
Initialise the algorithm

    1. Calculate for every item $s_i = \frac{p_i}{w_i}$ and rank them based on the score.
    2. Pick items starting from the highest rank until the knapsack is full.
    3. Set $LB$ = 0%
    4. Set $UB \in$ {0%, 100%}
    5. Set $\varepsilon$
    6. Set $Z^* = -inf$, the best improvement found so far.
    7. counter = 0

while $UB - LB > \varepsilon$:

    1. for $C \in \{UB, LB\}$
           Modify the distance matrix: $d'_{ij} = \begin{cases} (1 - C)\, d_{ij} \text{ if } i \in N_h \text{ and } j \in N_1 \\ (1 - C)\, d_{ij} \text{ if } i \in N_1 \text{ and } j \in N_1 \\ (1 + C)\, d_{ij} \text{ if } i \in N_1 \text{ and } j \in N_0 \\ (1 + C)\, d_{ij} \text{ if } i \in N_0 \text{ and } j \in N_1 \\ d_{ij} \text{ else} \end{cases}$
    2. Run LK algorithm with modified distances and evaluate the objective
       function (OBJ) for $LB$ and $UB$.
    3. if OBJ($UB$) $>$ OBJ($LB$)
           $Z^*$ = OBJ($UB$)
           $LB += (UB - LB) \cdot max(\frac{1}{2}, \frac{counter}{20})$
    4. else
           $Z^*$ = OBJ($LB$)
           $UB -= (UB - LB) \cdot max(\frac{1}{2}, \frac{counter}{20})$
    5. counter $+=$ 1

return $Z^*$
```

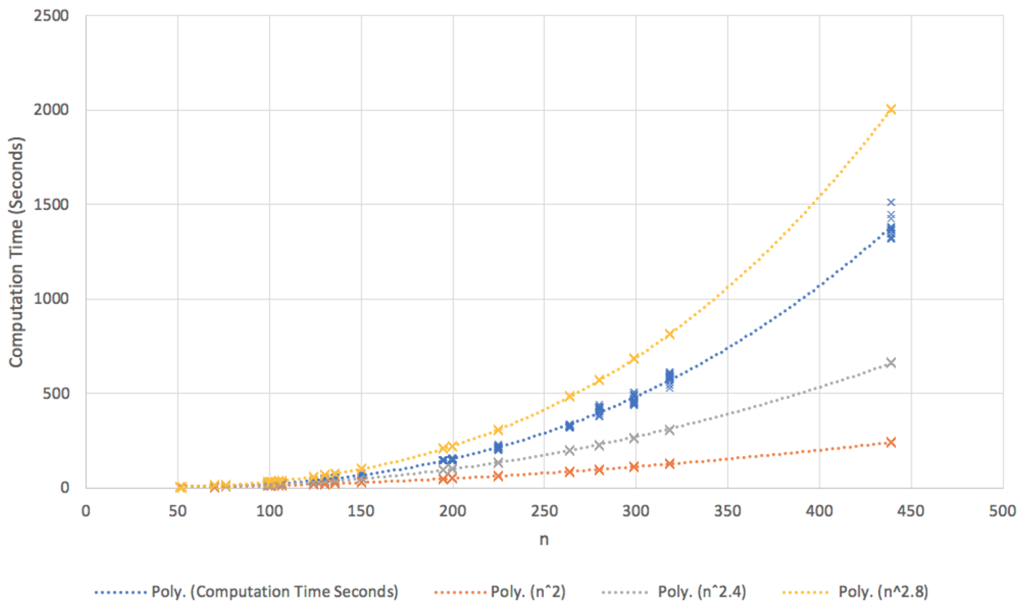
## 5 Results

### 5.1 Results for the Lin-Kernighan Algorithm

In Table 3, we can see some general information about the problems tested. We note that the amount of cities ranges from 51 to 439. For problems with a larger number of cities computation times became too large for the scope of this research. For every problem, 20 random starting tours were tried and every problem instance was run 10 times.

In an empirical analysis of the average computation times, we can see from figure 8 that it grows in polynomial time somewhere in between $n^{2.4}$ and $n^{2.8}$. This is more than Lin and Kernighan's implementation, but still a reasonable complexity. This discrepancy may have arisen from minor implementation differences. As, Laporte (2010) pointed out: "an efficient implementation of this algorithm requires a fair amount of sophistication regarding data structures and programming techniques, which makes it difficult to reproduce." So there might be a lot of speed to be gained by improving the programming efficiency. The algorithm is run on an MacBook Pro (2012), with an 2,6 GHz Intel Core i7 processor and an 8 GB RAM 1600 MHz DDR3 memory.

Figure 8: Growth of computation times of the Lin-Kernighan algorithm



In Table 4, we can see some of the performance statistics of the problem. Lin and Kernighan also tested the *kro{ABCDE}100* problems from Krolak et al. (1971) and found the optimum to be reached between 30% and 60% of the times. Our implementation does worse; the optimum is never reached and at best we often arrive at a solution within 10% of the optimum.

Table 3: Performance Lin-Kernighan algorithm (1)

| Problem Name | n | Optimal Value | Mean % From Optimum* | Mean Computation Time in Seconds* |
|---|---|---|---|---|
| *eil51* | 51 | 426 | 3,3% (0,3%) | 3,5 (0,1) |
| *berlin52* | 52 | 7542 | 4,2% (0,4%) | 4,3 (0,0) |
| *st70* | 70 | 675 | 6,8% (0,4%) | 8,4 (0,1) |
| *eil76* | 76 | 538 | 6,0% (0,3%) | 11,0 (0,1) |
| *pr76* | 76 | 108159 | 7,5% (0,2%) | 9,6 (0,1) |
| *rat99* | 99 | 1211 | 6,6% (0,4%) | 20,6 (0,1) |
| *kroA100* | 100 | 21282 | 8,9% (0,4%) | 21,2 (0,1) |
| *kroB100* | 100 | 22141 | 9,3% (0,4%) | 22,9 (0,2) |
| *kroC100* | 100 | 20749 | 10,9% (0,5%) | 19,8 (0,1) |
| *kroD100* | 100 | 21294 | 11,8% (0,4%) | 20,4 (0,1) |
| *kroE100* | 100 | 22068 | 7,3% (0,4%) | 21,2 (0,1) |
| *eil101* | 101 | 640,52 | 5,3% (0,2%) | 23,8 (0,1) |
| *lin105* | 105 | 14379 | 16,7% (0,8%) | 23,0 (0,1) |
| *pr107* | 107 | 44303 | 21,4% (0,8%) | 23,7 (0,2) |
| *pr124* | 124 | 59030 | 18,6% (0,8%) | 37,2 (0,5) |
| *ch130* | 130 | 6110 | 14,2% (0,6%) | 48,3 (0,4) |
| *pr136* | 136 | 96772 | 10,0% (0,3%) | 54,1 (1,4) |
| *ch150* | 150 | 6528 | 10,4% (0,4%) | 65,8 (0,9) |
| *kroA150* | 150 | 26524 | 14,5% (0,6%) | 65,3 (0,3) |
| *kroB150* | 150 | 26130 | 15,0% (0,6%) | 62,9 (0,2) |
| *rat195* | 195 | 2323 | 9,1% (0,3%) | 146,6 (0,6) |
| *kroA200* | 200 | 29368 | 18,1% (0,5%) | 149,9 (1,0) |
| *kroB200* | 200 | 29437 | 16,7% (0,4%) | 150,4 (0,7) |
| *tsp225* | 225 | 3919 | 13,4% (0,4%) | 212,7 (1,9) |
| *pr264* | 264 | 49135 | 30,1% (0,9%) | 326,9 (1,3) |
| *a280* | 280 | 2579 | 14,1% (0,7%) | 412,9 (4,1) |
| *pr299* | 299 | 48191 | 22,8% (0,7%) | 465,4 (5,0) |
| *lin318* | 318 | 42029 | 34,6% (0,7%) | 583,8 (4,7) |
| *pr439* | 439 | 107217 | 37,7% (1,0%) | 1381,0 (14,6) |

* standard errors in parentheses

Table 4: Performance Lin-Kernighan algorithm (2)

| Problem Name | n | % times: | Optimum | within 2% | within 5% | within 10% | within 20% | within 30% |
|---|---|---|---|---|---|---|---|---|
| *eil51* | 51 | | 0% | 20% | 95% | 100% | 100% | 100% |
| *berlin52* | 52 | | 0% | 15% | 60% | 100% | 100% | 100% |
| *st70* | 70 | | 0% | 0% | 15% | 100% | 100% | 100% |
| *eil76* | 76 | | 0% | 0% | 20% | 100% | 100% | 100% |
| *pr76* | 76 | | 0% | 0% | 0% | 100% | 100% | 100% |
| *rat99* | 99 | | 0% | 0% | 15% | 95% | 100% | 100% |
| *kroA100* | 100 | | 0% | 0% | 0% | 70% | 100% | 100% |
| *kroB100* | 100 | | 0% | 0% | 0% | 65% | 100% | 100% |
| *kroC100* | 100 | | 0% | 0% | 0% | 75% | 200% | 200% |
| *kroD100* | 100 | | 0% | 0% | 0% | 15% | 100% | 100% |
| *kroE100* | 100 | | 0% | 0% | 5% | 95% | 100% | 100% |
| *eil101* | 101 | | 0% | 0% | 50% | 100% | 100% | 100% |
| *lin105* | 105 | | 0% | 0% | 0% | 0% | 80% | 100% |
| *pr107* | 107 | | 0% | 0% | 0% | 0% | 40% | 100% |
| *pr124* | 124 | | 0% | 0% | 0% | 0% | 60% | 100% |
| *ch130* | 130 | | 0% | 0% | 0% | 0% | 100% | 100% |
| *pr136* | 136 | | 0% | 0% | 0% | 55% | 100% | 100% |
| *ch150* | 150 | | 0% | 0% | 0% | 40% | 100% | 100% |
| *kroA150* | 150 | | 0% | 0% | 0% | 10% | 100% | 100% |
| *kroB150* | 150 | | 0% | 0% | 0% | 5% | 100% | 100% |
| *rat195* | 195 | | 0% | 0% | 0% | 85% | 100% | 100% |
| *kroA200* | 200 | | 0% | 0% | 0% | 0% | 75% | 100% |
| *kroB200* | 200 | | 0% | 0% | 0% | 0% | 95% | 100% |
| *tsp225* | 225 | | 0% | 0% | 0% | 0% | 100% | 100% |
| *pr264* | 264 | | 0% | 0% | 0% | 0% | 0% | 50% |
| *a280* | 280 | | 0% | 0% | 0% | 10% | 100% | 100% |
| *pr299* | 299 | | 0% | 0% | 0% | 0% | 20% | 100% |
| *lin318* | 318 | | 0% | 0% | 0% | 0% | 0% | 10% |
| *pr439* | 439 | | 0% | 0% | 0% | 0% | 0% | 5% |

From table 3 we note that the average solution distance from the optimum grows with problem size. The larger the problem becomes, the worse the LK-algorithm performs at arriving at good solutions. As Helsgaun (2000) pointed out, for larger problems Lin and Kernighan's rule to restrict the search for a new $y_i$ to the five nearest neighbours doesn't work properly anymore.

Therefore we investigate whether changing this rule in the heuristic would lead to better results. As computation times rise significantly due to adding more $y_i$'s to be investigated, it is set as a variable number: $max(5, \frac{n}{5})$. This means that for a problem with 50 cities, 5 $y_i$'s are considered, but for example for a 150-city problem it will be 15. We reran the program for the *ch130*, *ch150*, the *kro{ABCDE}100* and the *kro{AB}150* problems.

Table 5: Results after considering more nearest neighbours

| Problem Name | n | Optimal Value | Previous % Mean From Optimum | Mean % From Optimum* | Mean Computation Time in Seconds* |
|---|---|---|---|---|---|
| *ch130* | 130 | 6110 | 14,2% (0,6%) | 10,9% (0,3%) | 272,8 (3,3) |
| *ch150* | 150 | 6528 | 10,4% (0,4%) | 7,2% (0,3%) | 459,2 (4,7) |
| *kroA100* | 100 | 21282 | 8,9% (0,4%) | 8,2% (0,3%) | 103,9 (0,9) |
| *kroB100* | 100 | 20749 | 9,3% (0,4%) | 7,5% (0,3%) | 112,2 (1,3) |
| *kroC100* | 100 | 20749 | 10,9% (0,5%) | 7,7% (0,3%) | 95,3 (0,9) |
| *kroD100* | 100 | 21294 | 11,8% (0,4%) | 7,5% (0,2%) | 104,0 (0,8) |
| *kroE100* | 100 | 22068 | 7,3% (0,4%) | 5,8% (0,3%) | 102,1 (1,1) |
| *kroA150* | 150 | 26524 | 14,5% (0,5%) | 9,4% (0,2%) | 485,9 (4,1) |
| *kroB150* | 150 | 26130 | 15,5% (0,6%) | 9,4% (0,4%) | 469,7 (3,6) |

* standard errors in parentheses

In table 5 we see that indeed by introducing this new rule, the average distance from the optimal value did decrease for these problem instances, most noticeably for the 150 city problems. However, there is the huge drawback in computation times. For the 100-city problems, the computation times grew with a factor 5, and for the 130-city and 150-city instances the computation times got 7 to 8 times larger. As for backtracking, we noticed very little effect in varying the amount of backtracking options considered at step 1 and 2.

## 5.2 Results for the LKMD Algorithm for the Travelling Thief Problem

The results for the LKMD algorithm can be found in table 6. There are a few interesting things to note about the results. Firstly we see that for almost all instances the LKMD outperforms the naive method of optimising both problems separately. There does not seem to be an apparent link between renting rate and outperformance or knapsack capacity and outperformance. The outperformance does appear to be strongest for the uncorrelated similar weights, and weakest for the uncorrelated case.

When comparing the results of the DH and SH algorithms in table 7, we don't note a large difference between the DH and SH algorithms. They are of course very similar. The LKMD algorithm outperforms both the DH and SH, but also has a running time that is sixty times larger. In the LKMD algorithm, the combination of a tour and packing routine are tried several times, whereas for the DH and SH algorithms it is tried only once. Its outperformance might therefore simply stem from the fact that the LKMD algorithm is repeated more often.

Comparing the results with the results of PACKITERATIVE in table 8 will be more comparable, as the PACKITERATIVE procedure also repeats the last step in the algorithm (finding a packing plan) several times. The PACKITERATIVE was run with variables $c = 5$, $\delta = 4$, $\varepsilon = 0.001$ and $q = 20$. On average, the LKMD outperforms the PACKITERATIVE. However, by inspecting the results closer, we noticed that for the PACKITERATIVE there are two cases. Either a packing plan is found, and the algorithm performs well, or no items are picked and the objective is equal to the tour length times the renting rate. The first conditional average value is found in the column 'filled' and the latter in the column 'empty'. When the PACKITERATIVE manages to find a packing plan for the tour, the algorithm performs better than the LKMD. Especially for problems the LKMD, DH and SH performs badly, such as *eil51_n50_bounded-strongly-corr_07* or *eil51_n50_uncorr-similar-weights_10*, the PACKITERATIVE finds good starting values.

Addionally, if one would run the PACKITERATIVE a certain number of times, one would become more sure that a filled solution would be found, which outperforms the LKMD algorithm. For example, for the *uncorr_02* instance, in 30% of the times a filled solution is found. If one would run the algorithm 9 times, the probability of finding a filled solution (and thus outperforming the LKMD algorithm) is $1-(1-0.3)^9 = 96\%$.

Table 6: LKMD results for the *eil51* instances

| File Name | Capacity | R | $Z^*$ | $Z^*_{\text{LKMD}}$ | $Z^*_{naive}$ | Mean distance from $Z^*$ | Improvement w.r.t NA | Mean Multiplier | Mean runtime |
|---|---|---|---|---|---|---|---|---|---|
| eil51_n50_bounded-strongly-corr_01 | 4029 | 4,44 | **4269** | 3932,5 (25,6) | 3350,3 (70,3) | 7,9% | 17,4% | 8,4 (0,9) | 316,3 (2,9) |
| eil51_n50_bounded-strongly-corr_02 | 8059 | 10,27 | **5571** | 3577,1 (23,2) | 2973,1 (110,8) | 35,8% | 20,3% | 8,7 (1,0) | 312,1 (1,8) |
| eil51_n50_bounded-strongly-corr_03 | 12089 | 16,95 | **5885** | 2665,3 (93,5) | 1855,8 (172,1) | 54,7% | 43,6% | 9,0 (0,9) | 308,8 (3,2) |
| eil51_n50_bounded-strongly-corr_04 | 16119 | 22,11 | **6310** | 2477,8 (123,7) | 2601,0 (79,0) | 60,7% | -4,7% | 9,7 (1,4) | 281,1 (2,9) |
| eil51_n50_bounded-strongly-corr_05 | 20149 | 29,77 | **4906** | 719,2 (148,2) | 611,6 (177,8) | 85,3% | 17,6% | 7,3 (0,6) | 286,0 (1,8) |
| eil51_n50_bounded-strongly-corr_06 | 24178 | 31,77 | **7083** | 3376,5 (179,7) | 2726,0 (329,9) | 52,3% | 23,9% | 9,1 (0,7) | 285,1 (1,2) |
| eil51_n50_bounded-strongly-corr_07 | 28208 | 34,68 | **8240** | 4200,1 (141,8) | 2769,5 (333,2) | 49,0% | 51,7% | 9,8 (0,8) | 287,1 (1,4) |
| eil51_n50_bounded-strongly-corr_08 | 32238 | 40,67 | **7059** | 1111,1 (236,2) | 242,5 (211,3) | 84,3% | 358,1% | 9,6 (0,5) | 255,2 (3,0) |
| eil51_n50_bounded-strongly-corr_09 | 36268 | 45,37 | **6775** | 3096,4 (161,7) | 2631,2 (271,9) | 54,3% | 17,7% | 10,1 (0,9) | 257,9 (1,3) |
| eil51_n50_bounded-strongly-corr_10 | 40298 | 43,49 | **11000** | 8456,0 (210,7) | 7755,2 (341,6) | 23,1% | 9,0% | 11,7 (0,8) | 264,5 (1,3) |
| eil51_n50_uncorr_01 | 2226 | 7,19 | **2851** | 2679,1 (23,9) | 2569,4 (58,9) | 6,0% | 4,3% | 8,1 (0,7) | 295,3 (3,6) |
| eil51_n50_uncorr_02 | 4452 | 9,72 | **4791** | 4452,0 (46,5) | 4007,9 (105,9) | 7,1% | 11,1% | 12,2 (0,7) | 287,7 (1,6) |
| eil51_n50_uncorr_03 | 6679 | 12,95 | **5404** | 4034,5 (52,5) | 4041,5 (67,3) | 25,3% | -0,2% | 9,7 (0,8) | 273,5 (3,7) |
| eil51_n50_uncorr_04 | 8905 | 19,66 | **3013** | 1340,0 (83,4) | 606,8 (155,1) | 55,5% | 120,8% | 9,3 (0,6) | 266,6 (3,7) |
| eil51_n50_uncorr_05 | 11132 | 19,6 | **4408** | 1498,7 (93,5) | 1228,0 (185,2) | 66,0% | 22,1% | 10,2 (0,9) | 276,8 (2,5) |
| eil51_n50_uncorr_06 | 13358 | 21,15 | **4440** | 2055,8 (92,1) | 1883,4 (275,2) | 53,7% | 9,2% | 8,7 (0,8) | 276,6 (2,2) |
| eil51_n50_uncorr_07 | 15585 | 23,03 | **4142** | 220,4 (106,8) | -287,3 (161,7) | 94,7% | 176,7% | 10,0 (1,2) | 290,7 (2,5) |
| eil51_n50_uncorr_08 | 17811 | 23,35 | **4790** | 1301,2 (79,3) | 1437,4 (136,4) | 72,8% | -9,5% | 10,3 (1,1) | 270,5 (1,4) |
| eil51_n50_uncorr_09 | 20038 | 22,52 | **6122** | 1740,5 (102,6) | 1650,4 (52,3) | 71,6% | 5,5% | 8,9 (1,4) | 262,4 (3,2) |
| eil51_n50_uncorr_10 | 22264 | 22,33 | **6821** | 1711,0 (99,2) | 1820,4 (134,2) | 74,9% | -6,0% | 9,6 (1,1) | 257,9 (2,4) |
| eil51_n50_uncorr-similar-weights_01 | 4567 | 3,82 | **1448** | 1192,6 (11,9) | 949,4 (43,4) | 17,6% | 25,6% | 8,3 (1,1) | 301,4 (1,4) |
| eil51_n50_uncorr-similar-weights_02 | 9135 | 5,83 | **3769** | 2724,0 (25,1) | 2071,3 (94,8) | 27,7% | 31,5% | 9,8 (1,0) | 291,2 (0,9) |
| eil51_n50_uncorr-similar-weights_03 | 13703 | 9,09 | **4433** | 3812,3 (36,7) | 3151,2 (95,5) | 14,0% | 21,0% | 12,1 (1,4) | 278,7 (1,4) |
| eil51_n50_uncorr-similar-weights_04 | 18270 | 14,47 | **3160** | 1831,4 (52,2) | 1122,0 (129,4) | 42,0% | 63,2% | 12,9 (1,2) | 264,4 (1,3) |
| eil51_n50_uncorr-similar-weights_05 | 22838 | 19,3 | **2134** | 740,9 (54,7) | 97,2 (127,5) | 65,3% | 662,5% | 10,1 (0,9) | 254,7 (1,0) |
| eil51_n50_uncorr-similar-weights_06 | 27406 | 21,21 | **2743** | 1102,1 (62,0) | 78,2 (172,1) | 59,8% | 1310,1% | 9,1 (0,5) | 246,8 (1,1) |
| eil51_n50_uncorr-similar-weights_07 | 31974 | 23,28 | **2857** | 212,7 (58,6) | 299,6 (119,9) | 92,6% | -29,0% | 9,8 (1,0) | 258,4 (3,2) |
| eil51_n50_uncorr-similar-weights_08 | 36541 | 24,12 | **3452** | 37,2 (37,2) | -398,8 (73,4) | 98,9% | 109,3% | 8,5 (1,0) | 266,1 (1,3) |
| eil51_n50_uncorr-similar-weights_09 | 41109 | 24,56 | **4094** | 111,4 (43,2) | 96,1 (84,2) | 97,3% | 15,8% | 9,3 (0,8) | 259,4 (3,2) |
| eil51_n50_uncorr-similar-weights_10 | 45677 | 23,78 | **5632** | 6,0 (6,0) | -51,4 (64,5) | 99,9% | 111,6% | 8,4 (1,2) | 256,6 (1,8) |

Standard errors of the mean in parentheses.

Table 7: LKMD compared to DH and SH

|  |  |  |  | **LKMD** |  | **DH** |  | **SH** |  |
|---|---|---|---|---|---|---|---|---|---|
| File Name | Capacity | R | $Z^*$ | $Z^*$ | runtime | $Z^*$ | runtime | $Z^*$ | runtime |
| eil51_n50_bounded-strongly-corr_01.ttp | 4029 | 4,44 | **4269** | 3932,5 (25,6) | 316,3 (2,9) | 1398,5 (646,1) | 5,0 (0,5) | 1408,1 (565,6) | 5,2 (0,8) |
| eil51_n50_bounded-strongly-corr_02.ttp | 8059 | 10,27 | **5571** | 3577,1 (23,2) | 312,1 (1,8) | 1668,2 (924,5) | 4,7 (0,2) | 1600,2 (572,2) | 4,6 (0,1) |
| eil51_n50_bounded-strongly-corr_03.ttp | 12089 | 16,95 | **5885** | 2665,3 (93,5) | 308,8 (3,2) | 3168,9 (729,5) | 4,7 (0,1) | 2378,7 (641,8) | 4,6 (0,2) |
| eil51_n50_bounded-strongly-corr_04.ttp | 16119 | 22,11 | **6310** | 2477,8 (123,7) | 281,1 (2,9) | 3687,8 (923,3) | 4,7 (0,2) | 3247,5 (1357,5) | 4,6 (0,2) |
| eil51_n50_bounded-strongly-corr_05.ttp | 20149 | 29,77 | **4906** | 719,2 (148,2) | 286,0 (1,8) | 1210,3 (1121,0) | 4,7 (0,2) | 2062,9 (795,9) | 4,7 (0,2) |
| eil51_n50_bounded-strongly-corr_06.ttp | 24178 | 31,77 | **7083** | 3376,5 (179,7) | 285,1 (1,2) | 2823,8 (3137,8) | 4,7 (0,1) | 3365,7 (2045,6) | 4,6 (0,2) |
| eil51_n50_bounded-strongly-corr_07.ttp | 28208 | 34,68 | **8240** | 4200,1 (141,8) | 287,1 (1,4) | 3682,0 (2854,4) | 4,7 (0,1) | 4880,2 (1583,2) | 4,7 (0,1) |
| eil51_n50_bounded-strongly-corr_08.ttp | 32238 | 40,67 | **7059** | 1111,1 (236,2) | 255,2 (3,0) | 814,9 (2752,6) | 4,7 (0,2) | -204,5 (4034,1) | 4,6 (0,2) |
| eil51_n50_bounded-strongly-corr_09.ttp | 36268 | 45,37 | **6775** | 3096,4 (161,7) | 257,9 (1,3) | -2959,1 (2171,5) | 4,8 (0,2) | -3518,5 (3256,8) | 4,6 (0,1) |
| eil51_n50_bounded-strongly-corr_10.ttp | 40298 | 43,49 | **11000** | 8456,0 (210,7) | 264,5 (1,3) | 3087,4 (2316,8) | 4,7 (0,1) | 2315,1 (3446,5) | 4,7 (0,2) |
| eil51_n50_uncorr_01.ttp | 2226 | 7,19 | **2851** | 2679,1 (23,9) | 295,3 (3,6) | -383,0 (425,6) | 4,7 (0,1) | -385,6 (492,0) | 4,8 (0,3) |
| eil51_n50_uncorr_02.ttp | 4452 | 9,72 | **4791** | 4452,0 (46,5) | 287,7 (1,6) | -290,7 (510,3) | 4,7 (0,1) | -287,3 (484,5) | 4,9 (0,2) |
| eil51_n50_uncorr_03.ttp | 6679 | 12,95 | **5404** | 4034,5 (52,5) | 273,5 (3,7) | -1202,1 (984,8) | 4,7 (0,1) | -1076,6 (801,7) | 5,3 (0,8) |
| eil51_n50_uncorr_04.ttp | 8905 | 19,66 | **3013** | 1340,0 (83,4) | 266,6 (3,7) | -5209,0 (1490,5) | 4,8 (0,2) | -4166,2 (715,0) | 5,1 (0,1) |
| eil51_n50_uncorr_05.ttp | 11132 | 19,6 | **4408** | 1498,7 (93,5) | 276,8 (2,5) | -2735,4 (1388,0) | 4,8 (0,1) | -2791,1 (1634,8) | 5,0 (0,1) |
| eil51_n50_uncorr_06.ttp | 13358 | 21,15 | **4440** | 2055,8 (92,1) | 276,6 (2,2) | -3445,1 (1841,1) | 4,7 (0,2) | -3463,7 (1977,3) | 5,0 (0,2) |
| eil51_n50_uncorr_07.ttp | 15585 | 23,03 | **4142** | 220,4 (106,8) | 290,7 (2,5) | -3291,7 (1068,1) | 4,8 (0,1) | -5000,2 (1436,2) | 4,9 (0,2) |
| eil51_n50_uncorr_08.ttp | 17811 | 23,35 | **4790** | 1301,2 (79,3) | 270,5 (1,4) | -3886,6 (2260,4) | 4,8 (0,1) | -3694,1 (1638,2) | 4,8 (0,2) |
| eil51_n50_uncorr_09.ttp | 20038 | 22,52 | **6122** | 1740,5 (102,6) | 262,4 (3,2) | -1968,7 (1817,8) | 4,8 (0,1) | -1498,8 (1740,1) | 4,8 (0,2) |
| eil51_n50_uncorr_10.ttp | 22264 | 22,33 | **6821** | 1711,0 (99,2) | 257,9 (2,4) | -1422,2 (1516,4) | 4,9 (0,1) | -973,6 (1445,8) | 5,0 (0,3) |
| eil51_n50_uncorr-similar-weights_01.ttp | 4567 | 3,82 | **1448** | 1192,6 (11,9) | 301,4 (1,4) | 983,2 (158,2) | 4,8 (0,2) | 1071,0 (119,2) | 4,8 (0,3) |
| eil51_n50_uncorr-similar-weights_02.ttp | 9135 | 5,83 | **3769** | 2724,0 (25,1) | 291,2 (0,9) | 2191,9 (547,6) | 4,8 (0,1) | 2343,9 (599,1) | 4,6 (0,2) |
| eil51_n50_uncorr-similar-weights_03.ttp | 13703 | 9,09 | **4433** | 3812,3 (36,7) | 278,7 (1,4) | 1921,6 (757,4) | 4,8 (0,1) | 2173,2 (483,2) | 5,1 (0,4) |
| eil51_n50_uncorr-similar-weights_04.ttp | 18270 | 14,47 | **3160** | 1831,4 (52,2) | 264,4 (1,3) | 76,5 (709,4) | 4,8 (0,2) | -384,8 (1021,8) | 4,8 (0,3) |
| eil51_n50_uncorr-similar-weights_05.ttp | 22838 | 19,3 | **2134** | 740,9 (54,7) | 254,7 (1,0) | -2540,1 (1337,6) | 4,7 (0,2) | -2539,6 (826,8) | 4,6 (0,2) |
| eil51_n50_uncorr-similar-weights_06.ttp | 27406 | 21,21 | **2743** | 1102,1 (62,0) | 246,8 (1,1) | -2655,1 (1373,2) | 4,8 (0,2) | -3893,9 (694,2) | 4,9 (0,2) |
| eil51_n50_uncorr-similar-weights_07.ttp | 31974 | 23,28 | **2857** | 212,7 (58,6) | 258,4 (3,2) | -3731,8 (1137,3) | 4,7 (0,1) | -3508,6 (1456,4) | 4,9 (0,2) |
| eil51_n50_uncorr-similar-weights_08.ttp | 36541 | 24,12 | **3452** | 37,2 (37,2) | 266,1 (1,3) | -5112,0 (634,7) | 4,7 (0,1) | -4010,6 (1170,3) | 4,9 (0,3) |
| eil51_n50_uncorr-similar-weights_09.ttp | 41109 | 24,56 | **4094** | 111,4 (43,2) | 259,4 (3,2) | -2722,1 (1641,7) | 5,0 (0,3) | -2418,8 (959,6) | 5,0 (0,2) |
| eil51_n50_uncorr-similar-weights_10.ttp | 45677 | 23,78 | **5632** | 6,0 (6,0) | 256,6 (1,8) | -2929,0 (717,9) | 4,7 (0,2) | -2118,8 (1628,7) | 4,8 (0,1) |

Standard errors of the mean in parentheses.

Table 8: LKMD compared to PACKITERATIVE

| | | | | LKMD | | PACKITERATIVE | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| File Name | Capacity | R | $Z^*$ | $Z^*$ | runtime | $Z^*$ | filled | empty | % | runtime |
| eil51_n50_bounded-strongly-corr_01.ttp | 4029 | 4,44 | 4269 | 3932,5 (25,6) | 316,3 (2,9) | 1541,9 (3018,8) | 3879,4 | -1964,5 | 0,6 | 4,9 (0,1) |
| eil51_n50_bounded-strongly-corr_02.ttp | 8059 | 10,27 | 5571 | 3577,1 (23,2) | 312,1 (1,8) | -556,8 (5091,7) | 5356,7 | -4499,1 | 0,4 | 4,6 (0,0) |
| eil51_n50_bounded-strongly-corr_03.ttp | 12089 | 16,95 | 5885 | 2665,3 (93,5) | 308,8 (3,2) | -251,4 (6261,1) | 4575,3 | -7491,4 | 0,6 | 4,7 (0,0) |
| eil51_n50_bounded-strongly-corr_04.ttp | 16119 | 22,11 | 6310 | 2477,8 (123,7) | 281,1 (2,9) | -5043,4 (7476,8) | 5789,6 | -9686,1 | 0,3 | 5,0 (0,1) |
| eil51_n50_bounded-strongly-corr_05.ttp | 20149 | 29,77 | 4906 | 719,2 (148,2) | 286,0 (1,8) | -4745,2 (8843,2) | 3627,0 | -13117,4 | 0,5 | 4,7 (0,1) |
| eil51_n50_bounded-strongly-corr_06.ttp | 24178 | 31,77 | 7083 | 3376,5 (179,7) | 285,1 (1,2) | -1197,1 (9286,2) | 5508,7 | -11255,7 | 0,6 | 5,0 (0,2) |
| eil51_n50_bounded-strongly-corr_07.ttp | 28208 | 34,68 | 8240 | 4200,1 (141,8) | 287,1 (1,4) | 2368,3 (6405,0) | 4320,4 | -15200,5 | 0,9 | 4,8 (0,1) |
| eil51_n50_bounded-strongly-corr_08.ttp | 32238 | 40,67 | 7059 | 1111,1 (236,2) | 255,2 (3,0) | 5060,1 (1422,6) | 5060,1 | - | 1 | 4,7 (0,0) |
| eil51_n50_bounded-strongly-corr_09.ttp | 36268 | 45,37 | 6775 | 3096,4 (161,7) | 257,9 (1,3) | -9853,2 (13107,9) | 5349,0 | -19987,9 | 0,4 | 4,7 (0,0) |
| eil51_n50_bounded-strongly-corr_10.ttp | 40298 | 43,49 | 11000 | 8456,0 (210,7) | 264,5 (1,3) | -5663,8 (12083,4) | 5670,7 | -13220,1 | 0,4 | 4,7 (0,1) |
| eil51_n50_uncorr_01.ttp | 2226 | 7,19 | 2851 | 2679,1 (23,9) | 295,3 (3,6) | 1468,7 (1664,2) | 1985,6 | -3183,5 | 0,9 | 4,7 (0,0) |
| eil51_n50_uncorr_02.ttp | 4452 | 9,72 | 4791 | 4452,0 (46,5) | 287,7 (1,6) | -1894,2 (3834,0) | 3659,4 | -4274,3 | 0,3 | 4,7 (0,0) |
| eil51_n50_uncorr_03.ttp | 6679 | 12,95 | 5404 | 4034,5 (52,5) | 273,5 (3,7) | 3388,8 (327,3) | 3388,8 | - | 1 | 4,6 (0,1) |
| eil51_n50_uncorr_04.ttp | 8905 | 19,66 | 3013 | 1340,0 (83,4) | 266,6 (3,7) | 436,2 (2056,2) | 1360,7 | -3261,9 | 0,8 | 4,7 (0,0) |
| eil51_n50_uncorr_05.ttp | 11132 | 19,6 | 4408 | 1498,7 (93,5) | 276,8 (2,5) | 301,1 (4907,8) | 3064,1 | -6145,9 | 0,7 | 4,6 (0,0) |
| eil51_n50_uncorr_06.ttp | 13358 | 21,15 | 4440 | 2055,8 (92,1) | 276,6 (2,2) | 2256,7 (4091,3) | 3542,1 | -9312,4 | 0,9 | 4,8 (0,1) |
| eil51_n50_uncorr_07.ttp | 15585 | 23,03 | 4142 | 220,4 (106,8) | 290,7 (2,5) | -344,8 (5580,7) | 3218,5 | -5689,8 | 0,6 | 4,7 (0,0) |
| eil51_n50_uncorr_08.ttp | 17811 | 23,35 | 4790 | 1301,2 (79,3) | 270,5 (1,4) | 185,1 (5944,1) | 3613,2 | -7813,9 | 0,7 | 4,7 (0,0) |
| eil51_n50_uncorr_09.ttp | 20038 | 22,52 | 6122 | 1740,5 (102,6) | 262,4 (3,2) | -406,2 (8256,1) | 5987,4 | -9996,7 | 0,6 | 4,6 (0,0) |
| eil51_n50_uncorr_10.ttp | 22264 | 22,33 | 6821 | 1711,0 (99,2) | 257,9 (2,4) | 4655,4 (5125,2) | 6274,8 | -9920,0 | 0,9 | 4,7 (0,0) |
| eil51_n50_uncorr-similar-weights_01.ttp | 4567 | 3,82 | 1448 | 1192,6 (11,9) | 301,4 (1,4) | -798,8 (1418,5) | 1255,8 | -1679,3 | 0,3 | 4,6 (0,0) |
| eil51_n50_uncorr-similar-weights_02.ttp | 9135 | 5,83 | 3769 | 2724,0 (25,1) | 291,2 (0,9) | -883,7 (2703,7) | 3029,2 | -2560,6 | 0,3 | 4,6 (0,0) |
| eil51_n50_uncorr-similar-weights_03.ttp | 13703 | 9,09 | 4433 | 3812,3 (36,7) | 278,7 (1,4) | 1240,1 (3616,7) | 3468,3 | -3959,1 | 0,7 | 4,7 (0,0) |
| eil51_n50_uncorr-similar-weights_04.ttp | 18270 | 14,47 | 3160 | 1831,4 (52,2) | 264,4 (1,3) | 1815,2 (1211,5) | 2032,6 | -141,8 | 0,9 | 4,7 (0,1) |
| eil51_n50_uncorr-similar-weights_05.ttp | 22838 | 19,3 | 2134 | 740,9 (54,7) | 254,7 (1,0) | -243,7 (2947,9) | 800,2 | -4419,5 | 0,8 | 4,7 (0,1) |
| eil51_n50_uncorr-similar-weights_06.ttp | 27406 | 21,21 | 2743 | 1102,1 (62,0) | 246,8 (1,1) | -1268,6 (1343,2) | 336,2 | -1956,4 | 0,3 | 4,6 (0,0) |
| eil51_n50_uncorr-similar-weights_07.ttp | 31974 | 23,28 | 2857 | 212,7 (58,6) | 258,4 (3,2) | 1515,6 (909,5) | 1515,6 | - | 1 | 4,7 (0,0) |
| eil51_n50_uncorr-similar-weights_08.ttp | 36541 | 24,12 | 3452 | 37,2 (37,2) | 266,1 (1,3) | -2410,9 (5713,5) | 1660,2 | -6482,0 | 0,5 | 4,7 (0,0) |
| eil51_n50_uncorr-similar-weights_09.ttp | 41109 | 24,56 | 4094 | 111,4 (43,2) | 259,4 (3,2) | -2264,9 (3247,0) | 143,4 | -3297,1 | 0,3 | 4,7 (0,0) |
| eil51_n50_uncorr-similar-weights_10.ttp | 45677 | 23,78 | 5632 | 6,0 (6,0) | 256,6 (1,8) | 3641,9 (4998,4) | 5212,4 | -10492,5 | 0,9 | 4,7 (0,1) |

Standard errors of the mean in parentheses.

## 6 Conclusion

In this thesis we have researched the Lin-Kernighan algorithm for the Travelling Salesman Problem.

In section 2, we have discussed the relevant literature on the Travelling Salesman Problem so far. Even after nearly hundred years of investigation into different solution methods for the Travelling Salesman Problem, according to most literature, the Lin-Kernighan algorithm remains one of the best TSP solvers available to date.

In section 3 we gave an explanation of the Lin-Kernighan algorithm as described in the original paper by Lin and Kernighan (1973), and summarised the subsequent literature on and improvements made to the Lin-Kernighan algorithm. In this section we also set out our experimental analysis in investigating the Lin-Kernighan algorithm.

For our implementation of the Lin-Kernighan algorithm, we find a larger growth of running times in between $n^{2.4}$ and $n^{2.8}$, contrary to the $n^{2.2}$ reported by Lin and Kernighan in the original paper and other sources that implemented the algorithm. This discrepancy may have arisen due to coding inefficiencies in our implementation of the algorithm.

As for the performance of the algorithm, our implementation also performs worse than it does for Lin and Kernighan. This might be caused by two things: the omission of the 4-opt non-sequential move check at the end of the algorithm, or discrepancies in the implementation. Even though the first may have some effect on solution quality, we expect the difference mostly to be caused by the latter.

In section 4 we have explained a relatively new benchmark problem: the Travelling Thief Problem. Along with a thorough explanation of the problem and the discussion of the comprehensive benchmark set that has been developed in the past few year, we also gave a full overview of the solution methods developed so far.

Additionally, we have looked at the TTP construction heuristics commonly used for solving TTP instances: the DH, SH and PACKITERATIVE algorithms. These algorithms have in common that first an optimal tour is sought, and afterwards the knapsack is filled. In this thesis, we have proposed a construction algorithm, the LKMD heuristic, that seeks to do the opposite. First, the knapsack is filled, and with that information a tour is constructed. In the construction of the tour, the lengths between the cities become pseudo-lengths depending on the picking plan. In our experimental analysis we compare our results with the DH, SH and PACKITERATIVE algorithms.

The LKMD algorithm outperforms the DH and SH construction algorithm, but also at the cost of a larger computation times. The LKMD does not perform better than the PACKITERATIVE algorithm.

We have found the order of first choosing the tour and then optimising the knapsack to perform better than the other way around. Additionally, the construction of a tour is more costly than that of a knapsack and therefore iteratively looking for a knapsack after the tours is faster than trying many tours for a certain knapsack plan.

However, the idea of first choosing the picking plan and constructing the tour afterwards, along with the use of a pseudo-distance, based on the picking plan is promising. The LKMD can be refined by implementing measures to decrease running time. Also, other forms of pseudo-distances, such as one that depends on the score of the items in the cities, may offer a better performance. It may also inspire other construction algorithms to try to exploit the interdependency between the two subproblems.

## 7 Bibliography

Applegate, D., Bixby, R., Cook, W., and Chvátal, V. (1998). On the solution of traveling salesman problems.

Applegate, D., Chvatal, V., and Cook, W. (1990). Data structures for the lin-kernighan heuristic. In *talk presented at the CRPC TSP Workshop*.

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

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

Berenguer, X. (1979). A characterization of linear admissible transformations for the m-travelling salesmen problem. *European Journal of Operational Research*, 3(3):232–238.

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

Bonomi, E. and Lutton, J.-L. (1984). The n-city travelling salesman problem: Statistical mechanics and the metropolis algorithm. *SIAM review*, 26(4):551–568.

Bonyadi, M. R., Michalewicz, Z., and Barone, L. (2013). The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In *2013 IEEE Congress on Evolutionary Computation*, pages 1037–1044. IEEE.

Bonyadi, M. R., Michalewicz, Z., Przybylek, M. R., and Wierzbicki, A. (2014). Socially inspired algorithms for the travelling thief problem. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 421–428. ACM.

Camerini, P. M., Fratta, L., and Maffioli, F. (1975). On improving relaxation methods by modified gradient techniques. In *Nondifferentiable optimization*, pages 26–34. Springer.

Chauhan, C., Gupta, R., and Pathak, K. (2012). Survey of methods of solving tsp along with its implementation using dynamic programming approach. *International Journal of Computer Applications*, 52(4).

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

Christofides, N. and Eilon, S. (1969). An algorithm for the vehicle-dispatching problem. *Journal of the Operational Research Society*, 20(3):309–318.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581.

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812.

Crowder, H. and Padberg, M. W. (1980). Solving large-scale symmetric travelling salesman problems to optimality. *Management Science*, 26(5):495–509.

Dantzig, G., Fulkerson, R., and Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410.

Desrochers, M. and Laporte, G. (1991). Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints. *Operations Research Letters*, 10(1):27–36.

Dorigo, M. and Gambardella, L. M. (1997). Ant colonies for the travelling salesman problem. *biosystems*, 43(2):73–81.

El Yafrani, M. and Ahiod, B. (2017). A local search based approach for solving the travelling thief problem: The pros and cons. *Applied Soft Computing*, 52:795–804.

El Yafrani, M. and Ahiod, B. (2018). Efficiently solving the traveling thief problem using hill climbing and simulated annealing. *Information Sciences*, 432:231–244.

Escario, J. B., Jimenez, J. F., and Giron-Sierra, J. M. (2015). Ant colony extended: experiments on the travelling salesman problem. *Expert Systems with Applications*, 42(1):390–410.

Faulkner, H., Polyakovskiy, S., Schultz, T., and Wagner, M. (2015). Approximate approaches to the traveling thief problem. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 385–392. ACM.

Glover, F. (1989). Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206.

Glover, F. (1990). Tabu search—part ii. *ORSA Journal on computing*, 2(1):4–32.

Goldberg, D. E. and Holland, J. H. (1988). Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99.

Golden, B. L. and Skiscim, C. C. (1986). Using simulated annealing to solve routing and location problems. *Naval Research Logistics Quarterly*, 33(2):261–279.

Grötschel, M. (1977). Polyedrische charakterisierungen kombinatorischer optimierungsprobleme.

Grötschel, M. and Holland, O. (1987). A cutting plane algorithm for minimum perfect 2-matchings. *Computing*, 39(4):327–344.

Hains, D., Whitley, D., and Howe, A. (2012). Improving lin-kernighan-helsgaun with crossover on clustered instances of the tsp. In *International Conference on Parallel Problem Solving from Nature*, pages 388–397. Springer.

Heile, F. (2017). Is the total number of particles in the universe stable over long periods of time?

Held, M. and Karp, R. M. (1962). A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210.

Held, M. and Karp, R. M. (1970). The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162.

Held, M. and Karp, R. M. (1971). The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical programming*, 1(1):6–25.

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

Helsgaun, K. (2009). General k-opt submoves for the lin–kernighan tsp heuristic. *Mathematical Programming Computation*, 1(2-3):119–163.

Helsgaun, K. (2011). Solving the bottleneck traveling salesman problem using the lin-kernighan-helsgaun algorithm. *Computer Science Research Report*, (143):1–45.

Helsgaun, K. (2014). Solving the equality generalized traveling salesman problem using the lin-kernighan-helsgaun algorithm. *Roskilde University*.

Helsgaun, K. (2017). An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems.

Helsgaun, K. (2018). Using popmusic for candidate set generation in the lin-kernighan-helsgaun tsp solver.

Holland, J. H. (1992). Genetic algorithms. *Scientific american*, 267(1):66–73.

Hong, S. (1973). *A linear programming approach for the travelling salesman problem.* PhD thesis, The Johns Hopkins University, Baltimore, MD. Unpublished.

Jati, G. K. et al. (2011). Evolutionary discrete firefly algorithm for travelling salesman problem. In *International Conference on Adaptive and Intelligent Systems*, pages 393–403. Springer.

Johnson, D. S. (1990). Local optimization and the traveling salesman problem. In *International Colloquium on Automata, Languages, and Programming*, pages 446–461. Springer.

Johnson, D. S. and McGeoch, L. A. (1995). The traveling salesman problem: A case study in local optimization.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer.

Kato, Y. and Yasuhara, M. (2000). Recovery of drawing order from single-stroke handwriting images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(9):938–949.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

Krolak, P., Felts, W., and Marble, G. (1971). A man-machine approach toward solving the traveling salesman problem. *Communications of the ACM*, 14(5):327–334.

Laporte, G. (1992). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247.

Laporte, G. (2010). A concise guide to the traveling salesman problem. *Journal of the Operational Research Society*, 61(1):35–40.

Laporte, G., Mercure, H., and Nobert, Y. (1987). Generalized travelling salesman problem through n sets of nodes: the asymmetrical case. *Discrete Applied Mathematics*, 18(2):185–197.

Larranaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., and Dizdarevic, S. (1999). Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13(2):129–170.

Lenstra, J., Desroches, M., Savelbergh, M., and Soumis, F. (1988). Vehicle routing with time windows: optimization and approximation. *Vehicle routing: Methods and studies*, pages 65–84.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269.

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

Lokin, F. (1979). Procedures for travelling salesman problems with additional constraints. *European Journal of Operational Research*, 3(2):135–141.

Mahi, M., Baykan, Ö. K., and Kodaz, H. (2015). A new hybrid method based on particle swarm optimization, ant colony optimization and 3-opt algorithms for traveling salesman problem. *Applied Soft Computing*, 30:484–490.

Mak, K.-T. and Morton, A. J. (1993). A modified lin-kernighan traveling-salesman heuristic. *Operations Research Letters*, 13(3):127–132.

Martin, O., Otto, S. W., and Felten, E. W. (1992). Large-step markov chains for the tsp incorporating local search heuristics. *Operations Research Letters*, 11(4):219–224.

Mei, Y., Li, X., and Yao, X. (2016). On investigation of interdependence between sub-problems of the travelling thief problem. *Soft Computing*, 20(1):157–172.

Menger, K. (1932). Das botenproblem. *Ergebnisse eines mathematischen kolloquiums*, 2:11–12.

Miliotis, P. (1976). Integer programming approaches to the travelling salesman problem. *Mathematical Programming*, 10(1):367–378.

Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329.

Odili, J. B. and Mohmad Kahar, M. N. (2016). Solving the traveling salesman's problem using the african buffalo optimization. *Computational intelligence and neuroscience*, 2016:3.

Ottmann, W. and Widmayer, P. (1990). Algorithmen und datenstrukturen. bi-wiss.

Ouaarab, A., Ahiod, B., and Yang, X.-S. (2014). Discrete cuckoo search algorithm for the travelling salesman problem. *Neural Computing and Applications*, 24(7):1659–1669.

Padberg, M. and Rinaldi, G. (1987). Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1):1–7.

Padberg, M. and Rinaldi, G. (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100.

Polyakovskiy, S., Bonyadi, M. R., Wagner, M., Michalewicz, Z., and Neumann, F. (2014). A comprehensive benchmark set and heuristics for the traveling thief problem. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 477–484. ACM.

Potvin, J.-Y. (1996). Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, 63(3):337–370.

Ratliff, H. D. and Rosenthal, A. S. (1983). Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. *Operations Research*, 31(3):507–521.

Reinelt, G. (1991). Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384.

Reinelt, G. (1994). *The traveling salesman: computational solutions for TSP applications*. Springer-Verlag.

Saji, Y. and Riffi, M. E. (2016). A novel discrete bat algorithm for solving the travelling salesman problem. *Neural Computing and Applications*, 27(7):1853–1866.

Taillard, É. D. and Helsgaun, K. (2019). Popmusic for the travelling salesman problem. *European Journal of Operational Research*, 272(2):420–429.

Wagner, M., Lindauer, M., Mısır, M., Nallaperuma, S., and Hutter, F. (2018). A case study of algorithm selection for the traveling thief problem. *Journal of Heuristics*, pages 1–26.

Wang, K.-P., Huang, L., Zhou, C.-G., and Pang, W. (2003). Particle swarm optimization for traveling salesman problem. In *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 03EX693)*, volume 3, pages 1583–1585. IEEE.

Wu, J., Wagner, M., Polyakovskiy, S., and Neumann, F. (2017). Exact approaches for the travelling thief problem. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pages 110–121. Springer.

Wu, J., Zhou, L., Du, Z., Lv, Y., et al. (2019). Mixed steepest descent algorithm for the traveling salesman problem and application in air logistics. *Transportation Research Part E: Logistics and Transportation Review*, 126:87–102.

Wuijts, R. H. (2018). Investigation of the traveling thief problem. Master's thesis, Utrecht University Department of Information and Computing Sciences.

# 8 Appendices

## 8.1 Tables

Table 9: Comments and origins of TSP test instances. Source: Reinelt (1991).

| Problem Name | Description |
|---|---|
| *a280* | Drilling problem (Ludwig) |
| *berlin52* | 52 location in Berlin, published by Grötschel (1977) |
| *ch130* | ’130 city problem’ (Churritz) |
| *ch150* | ’150 city problem’ (Churritz) |
| *eil101* | TSP instances of a vehicle dispatching problem by Christofides and Eilon (1969) |
| *eil51* | TSP instances of a vehicle dispatching problem by Christofides and Eilon (1969) |
| *eil76* | TSP instances of a vehicle dispatching problem by Christofides and Eilon (1969) |
| *kroA100* | Instance from Krolak et al. (1971) |
| *kroA150* | Instance from Krolak et al. (1971) |
| *kroA200* | Instance from Krolak et al. (1971) |
| *kroB100* | Instance from Krolak et al. (1971) |
| *kroB150* | Instance from Krolak et al. (1971) |
| *kroB200* | Instance from Krolak et al. (1971) |
| *kroC100* | Instance from Krolak et al. (1971) |
| *kroD100* | Instance from Krolak et al. (1971) |
| *kroE100* | Instance from Krolak et al. (1971) |
| *lin105* | The subproblem of *lin318* as described in Lin and Kernighan (1973) |
| *lin318* | The large problem described in Lin and Kernighan (1973) |
| *pr76* | Drilling problem obtain from Tektronics, inc. by Padberg and Rinaldi (1991). |
| *pr107* | Drilling problem obtain from Tektronics, inc. by Padberg and Rinaldi (1991). |
| *pr124* | Drilling problem obtain from Tektronics, inc. by Padberg and Rinaldi (1991). |
| *pr136* | Drilling problem obtain from Tektronics, inc. by Padberg and Rinaldi (1991). |
| *pr264* | Drilling problem obtain from Tektronics, inc. by Padberg and Rinaldi (1991). |
| *pr299* | Drilling problem obtain from Tektronics, inc. by Padberg and Rinaldi (1991). |
| *pr439* | Drilling problem obtain from Tektronics, inc. by Padberg and Rinaldi (1991). |
| *rat195* | Rattled grid (Pulleyblank) |
| *rat99* | Rattled grid (Pulleyblank) |
| *st70* | instance from Krolak et al. (1971) |
| *tsp225* | TSP instance constructed by Reinelt (1991) for the TSPlib. |

## 8.2 List of Supplementary Java Code Files

Table 10: Java Code list

| File Name | Description |
|---|---|
| City.java | The file containing the City class. |
| Edge.java | The file containing the Edge class. |
| Item.java | The file containing the Item class. |
| LKalgorithm.java | The file containing all algorithms and functions described in the thesis. |
| Statistics.java | The file containing the Statistics class. |
| Swap.java | The file containing the Swap class. |
| Tour.java | The file containing the Tour class. |