Eliminating Edges in TSP Instances

Rasmus Thomas Schroeder

Born 20th May 1989 in Münster, Germany

31st July 2014

Master's Thesis Mathematics

Advisor: Prof. Dr. Stefan Hougardy

Research Institute for Discrete Mathematics

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Contents

1 Preface

The traveling salesman problem, TSP for short, is the problem of finding the shortest tour visiting a number of cities and returning back to the origin. It is well known to be NP-hard [5]. The first great success in finding optimal tours was obtained by Dantzig, Folkerson and Johnson in 1954 [4], when they found an optimal tour for a 49-city problem in the USA. They did not only find the tour, they could in fact prove that it is optimal. For the next 60 years, many milestones concerning the TSP have been reached. In 2006, the largest instance for which a tour could be proven to be optimal was solved [1]. The instance has 85,900 points and was solved using the fastest TSP algorithm available - the *Concorde* algorithm designed by Applegate, Bixby, Chvátal and Cook [3]. For a state of the art overview of algorithms for the TSP problem see [2].

In this thesis we discuss an approach which eliminates edges of a TSP instance, leaving a subset of edges still containing all optimal tours. The proposed algorithm allows to reduce the number of edges of a 100,000 vertex instance leaving an average degree of around 6 on the remaining graph. The runtime needed on a single core is less than three days. This algorithm can be used as a preprocessing step for any other TSP algorithm. As stated in [6], the overall runtime to solve a TSP instance with more than 2000 vertices using the Concorde algorithm can be reduced by a factor of 11.

Sections 2 and 3 provide theorems needed to eliminate edges. They hold for any arbitrary symmetric TSP instance. The instance is not required to be metric. In Section 4 we discuss an implementation for TSP instances using the euclidean length function EUC_2D.

In Section 5 results for instances from the well known TSPLIB [8], as well as for some other large instances are presented.

1.1 Related Work

Jonker and Volgenant [7] introduced the notion of *nonoptimal* edges, which are edges not contained in any optimal tour. Based on 2-opt moves they were able to find some nonoptimal edges. In this thesis this notion is also used and is extended to sets of edges. The provided theorems of this thesis allow to identify significantly more nonoptimal edges than the method given in [7]. However, the approach given in this thesis does not yield reasonable runtimes on complete graphs. Some special cases of this approach designed for complete graphs are discussed in [6]. The provided theorems allow to eliminate less edges than the methods discussed in this thesis. However, they can be implemented in constant time per edge, yielding very small runtimes on a complete graph. Combining these two methods yields a very powerful algorithm to find nonoptimal edges.

2 Definitions and General Edge Elimination

2.1 Instances and Subinstances

An instance (V, E, l) of the symmetric TSP consists of a simple graph (V, E) where V is the set of vertices and E is the set of edges, together with a length function $l : E \to \mathbb{R}_+$. A *tour* T is the set of edges of a cycle that contains each vertex of the graph exactly once. Its length is defined as $l(T) := \sum_{e \in T} l(e)$. The set of all tours is denoted by $\mathcal{T}$. A tour $T \in \mathcal{T}$ is called *optimal* if no other tour for this instance has smaller length. The set of all optimal tours is denoted by $\mathcal{T}^*$. Finding an optimal TSP tour is a well known NP-hard problem [5].

The algorithm proposed in this thesis has a very local view, i.e. it is only working on very little windows within an instance. The definition of a subinstance, depicted in Figure 1, makes this more formal.

Definition 2.1. *Let (V', E', l') be a TSP instance and $B \subseteq V'$. Then (V', E', B, l') is called a* subinstance *with* border B. *We call (V', E', B, l') a subinstance of a TSP instance (V, E, l) if the following conditions hold.*

1. $V' \subseteq V$
2. E' *is the set of induced edges of* V'
3. $l' = l|_{E'}$
4. $B = \{v \in V' \mid \exists w \in V \setminus V' \; vw \in E\}$

For an instance (V, E, l) with $F \subseteq E$ we define the induced *subinstance as $(V_F, E_F, B, l|_{E_F})$, where V_F is the set of incident vertices of F, E_F is the set of induced edges of V_F and B is the border.*

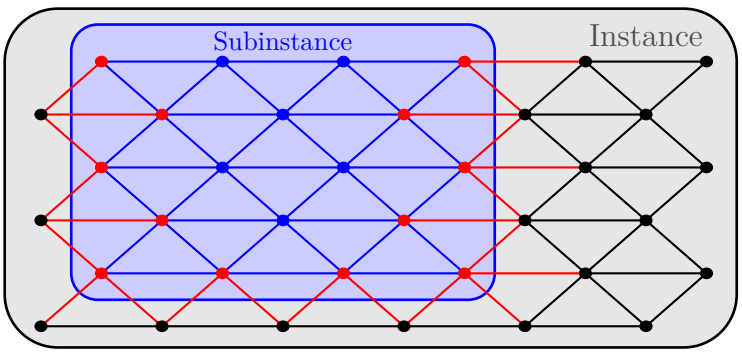

Figure 1: A TSP instance (V, E, l) with subinstance $(V', E', B, l_{E'})$. The edges E' are blue, the border B is red.

2.2 Nonoptimal Edges

Our goal is to find edges which are not part of any optimal tour. This leads us to the definition of *nonoptimal* edge sets.

Definition 2.2. *An edge set $F \subseteq E$ of a TSP instance (V, E, l) is called* optimal *if there exists an optimal tour $T \in \mathcal{T}^*$ with $F \subseteq T$. Otherwise F is called* nonoptimal. *In particular an edge e is called nonoptimal if it is not contained in any optimal tour.*

Lemma 2.3. *Given an instance (V, E, l) and $F \subseteq E$. Then it is NP-complete to decide whether F is nonoptimal.*

Proof. An instance (V, E) of Hamiltonian Cycle can be reduced to decide whether $F = \emptyset$ is optimal in (V, E, l) with $l \equiv 1$. □

For a given subinstance, there are different ways how it can be embedded in a TSP instance. However, we see later that it is possible to identify nonoptimal edges even if only a subinstance is known. Let us first introduce the concept of *local optimality*.

Definition 2.4. *Let (V', E', B, l') be a subinstance and $F \subseteq E'$. The set F is called* locally optimal *with respect to the subinstance, if there exists an instance (V, E, l) with (V', E', B, l') as a subinstance, for which there exists a tour $T \in \mathcal{T}^*$ with $F \subseteq T$. The set F is called* locally nonoptimal *with respect to the subinstance, if it is nonoptimal for every instance (V, E, l) for which (V', E', B, l') is a subinstance.*

By the definition we immediately obtain the following corollary.

Corollary 2.5. *Let (V, E, l) be a TSP instance with $(V', E', B, l|_{E'})$ as a subinstance. Let $F \subseteq E'$ be locally nonoptimal with respect to the subinstance. Then F is nonoptimal for the instance (V, E, l).*

The union of two optimal edge sets must not be optimal. For this we introduce the notion of *incompatible* edge sets.

Definition 2.6. *Let (V, E, l) be a TSP instance and $F_1, F_2 \subseteq E$ two edge sets. The sets F_1 and F_2 are called* compatible, *if $F_1 \cup F_2$ is optimal. Otherwise they are called* incompatible. *Further F_1 and F_2 are called* locally *compatible, denoted by $F_1 \sim F_2$, if $F_1 \cup F_2$ is locally optimal for the induced subinstance. Otherwise they are called* locally *incompatible, denoted by $F_1 \nsim F_2$.*

2.3 Coverings

In order to find nonoptimal edges, we introduce the concept of *coverings* for edge sets. Intuitively speaking, a covering of the edge set F contains all possibilities, how an optimal tour containing F can look like in the near surrounding of F.

Definition 2.7. *Let (V, E, l) be a TSP instance and $F \subseteq E$ a set of edges. A set of edge sets $\mathcal{C}_F \subseteq \mathcal{P}(E)$ is called a* covering *for F if the following conditions hold.*

- ***Subset condition***
  $\forall S \in \mathcal{C}_F \quad F \subseteq S$
- ***Tourcompatibility***
  $\forall T \in \mathcal{T}^* \ (F \subseteq T \Rightarrow \exists S \in \mathcal{C}_F \quad S \subseteq T)$

Note that a covering of $F = \emptyset$ is a collection of edge sets with the property that every optimal tour fully contains at least one of the sets. We call this special case of a covering an *optimal path system.*

Note that for every edge set F there always exists a covering $\mathcal{C}_F = \{F\}$ only containing the set itself.

As we are aiming towards working on subinstances only, the definition of a covering can be extended.

Definition 2.8. *Let (V', E', B, l') be a subinstance and $F \subseteq E'$ a set of edges. A set of edge sets $\mathcal{C}_F \subseteq \mathcal{P}(E')$ is called a* covering *for F if the following two conditions hold.*

- $\forall S \in \mathcal{C}_F \quad F \subseteq S$
- *For every instance (V, E, l) with (V', E', B, l') as a subinstance, the following holds*
  $\forall T \in \mathcal{T}^* \ (F \subseteq T \Rightarrow \exists S \in \mathcal{C}_F \quad S \subseteq T)$

The context makes clear, which definition of covering has to be used. We are using two basic covering operations, which allow to extend a given covering. The first operation allows to leave out nonoptimal elements.

Lemma 2.9. *Let (V, E, l) be a TSP instance, $F \subseteq E$, $\mathcal{C}_F$ a covering of F and $S \in \mathcal{C}_F$. If S is nonoptimal then $\mathcal{C}_F \setminus \{S\}$ is a covering of F.*

Proof. The subset condition of Definition 2.7 remains true after eliminating elements of $\mathcal{C}_F$. The tourcompatibility can only be violated, if there exists an optimal tour T with $S \subseteq T$. This contradicts the nonoptimality of S. □

The second covering operation allows to replace an element by a covering of this element.

Lemma 2.10. *Let (V, E, l) be a TSP instance, $F \subseteq E$, $\mathcal{C}_F$ a covering of F and $S \in \mathcal{C}_F$. Let $\mathcal{C}_S$ be a covering of S. Then $(\mathcal{C}_F \setminus \{S\}) \cup \mathcal{C}_S$ is a covering of F.*

Proof. To show the subset condition let $S' \in (\mathcal{C}_F \setminus \{S\}) \cup \mathcal{C}_S$. If $S' \in \mathcal{C}_F$, we have $F \subseteq S'$, since $\mathcal{C}_F$ is a covering of F. Otherwise if $S' \in \mathcal{C}_S$, then $F \subseteq S \subseteq S'$.

To show tourcompatibility let $T \in \mathcal{T}^*$ and $F \subseteq T$. In case $\exists S' \in \mathcal{C}_F \setminus \{S\}$ $S' \subseteq T$ nothing is to be shown. Otherwise we have that $S \subseteq T$. It is left to prove that $\exists S' \in \mathcal{C}_S$ with $S' \subseteq T$. By the definition of a covering we have that $F \subseteq S \subseteq T$. The proof now follows from the tourcompatibility of the covering $\mathcal{C}_S$. □

In fact Lemma 2.10 implies Lemma 2.9, since the empty set is a covering of every nonoptimal edge set.

2.4 Eliminating Edges

The concept of coverings already allows to prove a theorem showing nonoptimality for edge sets.

Theorem 2.11. *Given a TSP instance (V, E, l). If $\mathcal{C}_F := \emptyset$ is a covering of the edge set $F \subseteq E$, then F is nonoptimal.*

Proof. Assume for contradiction, that $F \subseteq T$ for an optimal tour T. By the tourcompatibility of $\mathcal{C}_F$ it follows that

$$\exists S \in \mathcal{C}_F \quad S \subseteq T.$$

This is a contradiction since $\mathcal{C}_F$ is empty. □

Another method to show nonoptimality is given in the following theorem.

Theorem 2.12. *Let (V, E, l) be a TSP instance, $\mathcal{C}_\emptyset$ an optimal path system and $F \subseteq E$. If for all $C \in \mathcal{C}_\emptyset$ we have $F \nsim C$, then the set F is nonoptimal.*

Proof. Assume for contradiction that $T \in \mathcal{T}^*$ and $F \subseteq T$. Tourcompatibility implies

$$\exists S \in \mathcal{C}_\emptyset \; S \subseteq T.$$

Hence $S \sim F$ contradicting the assumption of the theorem. □

In order to eliminate edges we only use Theorems 2.11 and 2.12. To be able to apply these theorems in practice, one has to find *good* coverings in a sense, that the conditions of these theorems are satisfied.

3 Finding Coverings

In this section we develop methods to generate *good* coverings. In Section 3.1 we deal with the extension of coverings. Deleting nonoptimal elements from a covering is done in Section 3.2.

3.1 Extending Coverings

The operation introduced in Lemma 2.10 extends a covering by replacing one of its elements by a covering of this element. In the following we see how a nontrivial covering of an edge set F can be found, which does not contain F as an element.

We make use of the fact that every tour has exactly two incident edges with every vertex.

Lemma 3.1. *Let (V, E, l) be a TSP instance, $F \subset E$ a set of edges and $v \in V$ with at most one incident edge in F. Let $\{e_1, \dots, e_k\}$ be the set of outgoing edges of v. Then*

$$\mathcal{C}_F := \{F \cup \{e_i, e_j\} \mid 1 \leq i < j \leq k\}$$

is a covering of F.

Proof. The subset condition is trivial. For the tourcompatibility let $T \in \mathcal{T}^*$ be an optimal tour with $F \subseteq T$. Let $e, f \in T$ be the edges of T incident to v. By the definition of $\mathcal{C}_F$ we conclude

$$F \cup \{e, f\} \in \mathcal{C}_F. \qquad \square$$

3.2 Proving Nonoptimality

This section deals with the core of the edge elimination given in this thesis. We develop methods showing that an edge set is nonoptimal. A rather trivial way to show nonoptimality for a set F is to show that the set cannot be part of any tour at all, which is discussed in Section 3.2.1. Otherwise, and if F is not a tour itself, it can be written as a set of paths. In Section 3.2.2 we see how a single k-opt move can be used to show nonoptimality. This method is generalized in Section 3.2.3. A method using different k-opt moves for different embeddings of F in an optimal tour is introduced. Note that everything discussed in Section 3.2.2 is a special case of the *path elimination* discussed in Section 3.2.3. However, for the implementation discussed in Section 4, these methods turn out to reduce practical running time.

3.2.1 Tour Conditions

The simplest way to show nonoptimality of an edge set is to show that it cannot be part of any tour at all. Let $F \subseteq E$ be the given set. The following statements are fairly obvious criterias:

- **Degree constraint:** There exists a vertex incident to at least three edges of F
- **Cycle constraint:** F contains a cycle with less than $|V|$ edges.

3.2.2 Path Preserving Conditions

We now assume that the set F does not violate the conditions of the previous section. Further we assume that $|F| < |V|$. Hence it can be written as a collection of paths. As shown in Figure 2 there exist natural numbers P (number of paths) and p_i for $i \leq P$ (length of path i) such that the incident vertices of F can be subscribed as

$$\begin{gathered} v_{1,0}, v_{1,1}, \ldots, v_{1,p_1} \\ v_{2,0}, v_{2,1}, \ldots, v_{2,p_2} \\ \vdots \\ v_{P,0}, v_{P,1}, \ldots, v_{P,p_P} \end{gathered}$$

and the edges are

$$\{(v_{i,j}, v_{i,j+1}) \mid 1 \leq i \leq P,\ 0 \leq j < p_i\}.$$

We refer to this as the *standard* subscription.

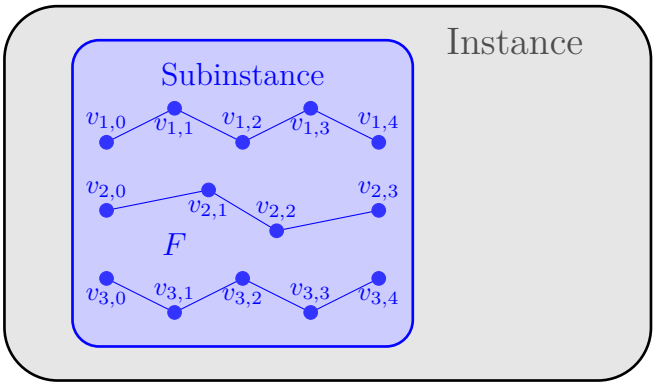

Figure 2: Induced subinstance $(V_F, E_F, B, l|_{E_F})$ with edge set F.

Let $(V_F, E_F, B, l|_{E_F})$ be the induced subinstance of F. Note that E_F can contain nonoptimal edges. As depicted in Figure 3, there are different possibilities, how the set F can be embedded in a tour.

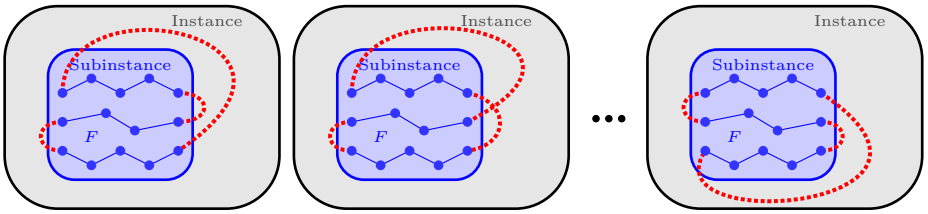

Figure 3: Different valid embeddings of F.

Definition 3.2. *Let (V_F, E_F, B, l) be the induced subinstance of an edge set F consisting of P paths. Let V_F have the standard subscription. Every maximum matching C in the complete graph on the vertices $\{v_{i,0} | i \leq P\} \cup \{v_{i,p_i} | i \leq P\}$ is called an* embedding *of the paths in F. An embedding C is called* valid*, if $C \cup F$ is a cycle. The set of all valid embeddings of F is denoted by $\mathcal{E}_F$.*

We want to use a single k-opt move to prove that the set F is nonoptimal. It is important to guarantee that the opt-move is valid, i.e. the resulting edge set is a tour. This has to hold for any optimal tour containing F. The following theorem states a possibility to guarantee an opt-move to be valid.

Theorem 3.3. *Let (V, E, l) be a TSP instance and $F \subset E$ be a collection of paths with standard subscription. Let $(V_F, E_F, B, l|_{E_F})$ be the induced subinstance of F. Further let $F' \subset E_F$ be a set with the following properties.*

1. *$|F'| = |F|$*
2. *F' is a collection of paths with the same endpoints as F, i.e. the i-th path has the end vertices $v_{i,0}$ and v_{i,p_i}*
3. *$l(F') < l(F)$*

Then F is nonoptimal.

Proof. Assume for contradiction, that $F \subset T$ with $T \in \mathcal{T}^*$. Define $T' = (T \setminus F) \cup F'$. We claim that T' is a tour of shorter length. By the first condition we have $|T| = |T'|$. The second condition forces T' to be a tour. And finally from the third condition we conclude that $l(T') < l(T)$. This contradicts the optimality of T. □

An example of Theorem 3.3 is depicted in Figure 4. In practice, a special case of these path preserving opt-moves turns out to be very helpful. It is a restriction to $k = 3$ as shown in Figure 5.

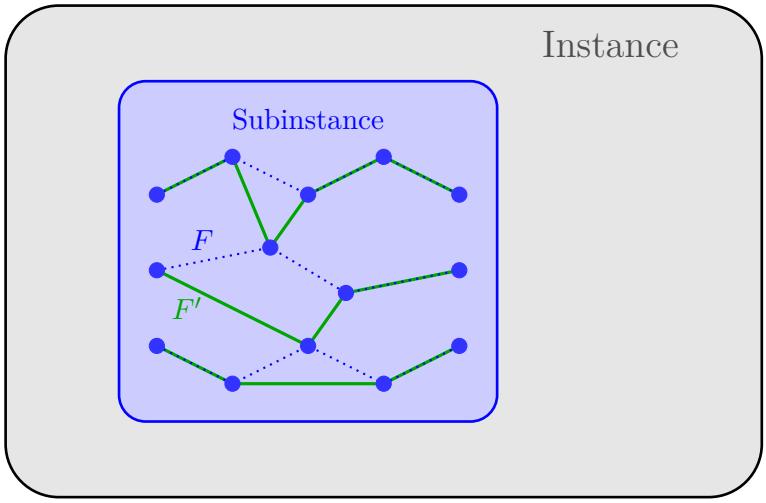

Figure 4: A path preserving opt move.

Corollary 3.4. *Let (V, E, l) be a TSP instance. Let $p, q, m, s, t \in V$ and $pq, pm, qm, ms, mt, st \in E$. If*

$$l(pm) + l(qm) + l(st) \;<\; l(pq) + l(ms) + l(mt)$$

the set $\{pq, ms, mt\}$ is nonoptimal.

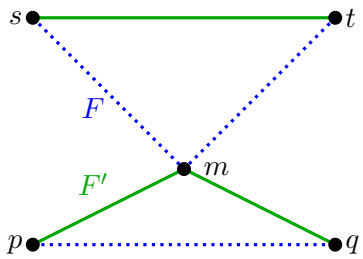

Figure 5: 3-opt move.

3.2.3 General Path Elimination

A more general way to prove nonoptimality is to drop the condition of preserving the paths. It is important to guarantee that the opt move is valid. This highly depends on how the set F is embedded in the instance. The trick is not to use one opt move only. One rather uses a different opt move for every possibility how the paths can be extended to a tour within

the instance. If all of these opt moves shorten the length, the set F is nonoptimal, as for every tour there exists at least one valid opt move. This is depicted in Figure 6.

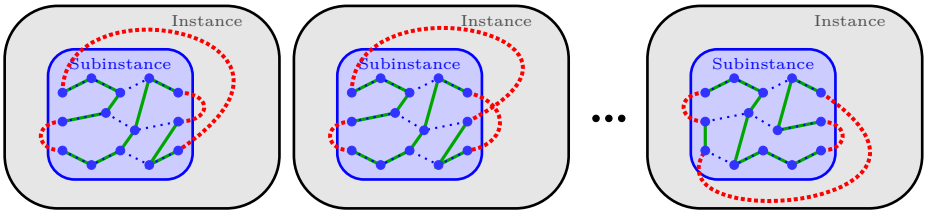

Figure 6: For every valid embedding of F, the green edges F' yield a valid opt move.

Theorem 3.5. *Let (V, E, l) be a TSP instance and $F \subset E$ a collection of paths with standard subscription. Let $(V_F, E_F, B, l|_{E_F})$ be the induced subinstance of F. Assume that for each $C \in \mathcal{E}_F$ there exists a set $F' \subseteq E_F$ with the following properties.*

1. *$C \cup F'$ is a cycle*
2. *$l(F') < l(F)$*

Then F is nonoptimal.

Proof. Assume for contradiction that $F \subseteq T$ with $T \in \mathcal{T}^*$. Let C be the embedding of F corresponding to T, i.e. a valid embedding with the property that the paths in F occur in the cycle $C \cup F$ in the same order and with the same orientations as they do in the tour T. By the conditions of the theorem there exists $F' \subseteq E_F$ such that $(T \setminus F) \cup F'$ is a tour with shorter length. This contradicts the optimality of T. □

The proof shows that it is not necessary to consider every tour containing a set F. One rather classifies all tours in a sense of which order and orientation the paths appear in the tour. The number of embeddings as defined in 3.2 is stated in the following lemma.

Lemma 3.6. *A set F of P paths has*

$$\frac{1}{P!}\prod_{i=1}^{P}\binom{2i}{2} \tag{1}$$

embeddings from which

$$\prod_{i=1}^{P-1}(2P-2i) \tag{2}$$

are valid.

Proof. The number of embeddings of P paths corresponds to the number of perfect matchings on $2P$ vertices. There are $\binom{2i}{2}$ possibilities to choose an edge of a complete graph with $2i$ vertices. When this is done repeatedly on a $2P$ vertex graph and eliminating the two chosen vertices, this yields $\prod_{i=1}^{P}\binom{2i}{2}$ possibilities, when the order of the choices is counted. Ignoring order yields (1).

The number of valid embeddings of P paths can be counted as follows. Starting at vertex $v_{1,0}$ one constructs a cycle containing all paths. For this one repeatedly chooses an edge connecting the so far constructed part of the cycle with one of the endpoints which is not yet in the cycle. When i paths are in the cycle already, there are $2P-2i$ such choices. The cycle is then continued at the other end of the just inserted path. The last endpoint to be chosen is v_{1,p_1}. In total (2) gives the number of valid embeddings. □

The following table states the number of embeddings for sets with up to 9 paths.

# paths	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
# embeddings	1	3	15	105	945	10395	135135	2027025	34459425
# valid embeddings	1	2	8	48	384	3840	46080	645120	10321920

To find an opt move for a given embedding, one can solve a smaller TSP instance. The set of matching edges can be fixed. Further one inserts all edges E'. The costs for the edges E' are set to the same as in the given instance. The length of the matching edges can be set to $-(l(E')+1)$, since then this is equivalent to forcing these edges, as shown in Lemma 3.7. It now suffices to solve this small TSP instance.

Lemma 3.7. *Let (V, E, l) be a TSP instance, $F \subset E$, and $M := l(E \setminus F)+1$. Further let l satisfy the following conditions*

$$l(e) \begin{cases} \geq 0 & \text{for } e \notin F \\ = -M & \text{for } e \in F \ . \end{cases}$$

If there exists a tour containing F, then any optimal tour also contains F.

Proof. Assume that $T \in \mathcal{T}^*$ with $e \in F \setminus T$. Let $T' \in \mathcal{T}$ be a tour with $F \subseteq T'$. Then

$$\begin{aligned} l(T') &= l(T' \setminus F) + l(F) \\ &\leq l(E \setminus F) - |F| \cdot M \\ &< l(E \setminus F) + 1 - |F| \cdot M \\ &= (1 - |F|) \cdot M \\ &\leq l(T) \end{aligned}$$

□

In order to prove that a set F is nonoptimal, for every valid embedding of F one has to solve a small TSP instance as shown above. In fact not all valid embeddings have to be considered. Some opt moves yield tours for different embeddings.

4 Edge Elimination Algorithm

The methods introduced in Sections 2 and 3 are now put together to an algorithm which eliminates edges of a TSP instance. The methods described hold for any arbitrary symmetric length function. They do not require the instance to be euclidean or metric.

In this section we discuss an implementation designed to work on graphs using the EUC_2D length function. This is the most common length function used in the TSPLIB [8]. The vertices are points in the 2-dimensional plane, given by their coordinates. The length between two vertices is the euclidean distance between the points, rounded to the nearest integer. Note that because of the rounding the length function is not metric. The graph is assumed to be complete. Keeping in mind that the edge set of every instance contains every edge of the complete graph, we denote by (V, E, l) an instance, where all optimal tours are contained in E. The algorithm runs in three steps.

4.1 Step 1: Elimination on a Complete Graph

A direct implementation of the methods discussed in this thesis will not be practical for large complete graphs. Finding reasonable coverings will cause huge runtime problems. A special case of these methods which can eliminate an edge in constant time is discussed in [6].

4.2 Step 2: Special Cases on a Sparse Graph

This step restricts its elimination by using coverings with five edges only. This restriction allows a faster implementation than the general case described in Step 3. It turns out to eliminate a significant number of edges. The pseudocode is given in Algorithm 1.

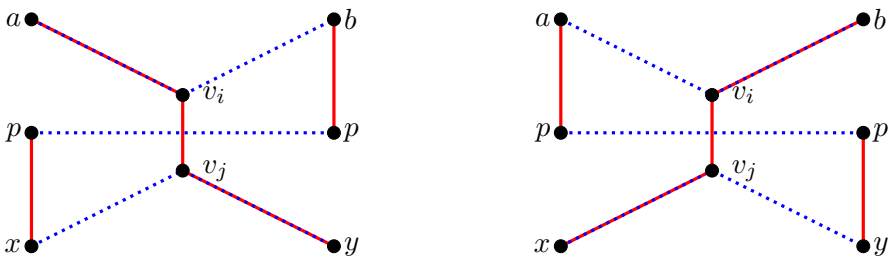

Figure 7: Two opt moves for an element of $\mathcal{C}_{pq}$ in Step 2

In line 12 the set $\mathcal{C}_{pq}$ is a covering of the edge pq. It can be obtained in the following way. Start with the covering containing the edge pq only. Using

Algorithm 1 5-OptElimination

Input: An instance (V, E, l) containing all optimal tours, $c \in \mathbb{Z}_+$.
Output: An edge set $E' \subseteq E$ containing all optimal tours.

```
1: Set E' := E
2: for all pq ∈ E do
3:     Select v_1, ..., v_c ∈ V \ {p, q}
4:     For i ≤ c let N_{v_i} := {w ∈ V | v_i w ∈ E'}
5:     for all v_i, v_j ∈ {v_1, ..., v_c} with v_i v_j ⇝ pq do
6:         Set covering C_pq := ∅
7:         for all a, b ∈ N_{v_i} and x, y ∈ N_{v_j} do
8:             if {pq, v_i a, v_i b, v_j x, v_j y} is locally optimal then
9:                 C_pq ← C_pq ∪ {{pq, v_i a, v_i b, v_j x, v_j y}}
10:            end if
11:        end for
12:        if C_pq = ∅ then
13:            E' ← E' \ {pq}
14:        end if
15:    end for
16: end for
17: return E'
```

Lemma 3.1 extend the covering at node v_i from line 5. For all sets in the covering, again extend the covering at node v_j. Line 8 corresponds to the elimination of a set from the covering if it is nonoptimal as in Lemma 2.9. Hence the covering obtained in line 12 is indeed a covering of pq. By Theorem 2.11 the edge pq is correctly eliminated in line 13. Since line 5 forces $v_i v_j \nsim pq$, every element in the covering $\mathcal{C}_{pq}$ has the structure as depicted in Figure 7. In this special case of the path elimination, it suffices to use two 3-opt moves to guarantee a valid opt move for every embedding. In line 8 only these two opt moves must be checked. This can be done for every order of a, b and x, y respectively. This operation corresponds to the *Main Edge Elimination Theorem* in [6]. Corollary 3.4 provides an easy to check method to reduce the size of the covering $\mathcal{C}_{pq}$. In line 7 one can ignore a pair of neighbors a and b if the edge set containing pq, $v_i a$ and $v_i b$ is nonoptimal. Analogously for the set N_{v_j}.

In practice it turns out, that good candidate nodes chosen in line 3 are the ones close to the center of the line segment pq.

4.3 Step 3: Path Elimination

This step of the algorithm is designed to work on a sparse graph, as it highly depends on the degree of the vertices. Intuitively this step can eliminate edges which are *close to* a given vertex. The idea is to generate an optimal path system, i.e. a covering for the empty set, for which we know that every optimal tour contains at least one of the generated sets of paths. We can then use Theorem 2.12 to eliminate edges.

A crucial part of the algorithm is to show that a given edge set is non-optimal. Algorithm 2 is a subroutine which decides whether a given set of edges is locally optimal.

Algorithm 2 LocalOptimality

Input: An induced subinstance (V_F, E_F, B, l) of an edge set F.
Output: true if F is locally optimal, false if F is locally nonoptimal.

```
1:  Initialize E := E_F as the set of all valid embeddings of F
2:  while E ≠ ∅ do
3:      Let C ∈ E
4:      Solve (V_F, E_F ∪ C, l) with l(e) := −(l(E_F) + 1) if e ∈ C
5:      Let T be the tour
6:      if C ⊂ T and l(T) − l(C) ≥ l(F) then
7:          return true
8:      end if
9:      for all C′ ∈ E do
10:         if (T \ C) ∪ C′ is a tour then
11:             E ← E \ {C′}
12:         end if
13:     end for
14: end while
15: return false
```

In line 4 a TSP instance has to be solved. This seems to be an ambitous step in an algorithm trying to solve a TSP instance. However, in practice small instances can be solved very fast. One can for example use the Concorde algorithm to do this step. The edges of the induced subinstance from the input can be chosen as the complete edge set on V_F. Note that the tour in line 5 can contain nonoptimal edges. This does not matter, since an opt move can shorten a tour, even if some nonoptimal edges are used in the new tour.

Using this very essential subroutine we can now state an algorithm which can eliminate edges. This is done in Algorithm 3. The practical runtime of the algorithm depends a lot on the choice of the next selected vertex v in line 7. Choosing v to lie close to the center c increases the chance, that

Algorithm 3 PathElimination

Input: Subinstance (V, E, B, l), vertex $c \in V$, timebound t.
Output: Set of nonoptimal edges $H \subset E$.

1: Set covering to $\mathcal{C}_\emptyset := \{\emptyset\}$
2: Set selected vertices $\mathcal{S} : \mathcal{C}_\emptyset \to \mathcal{P}(V)$ to $\mathcal{S}(\emptyset) := \emptyset$
3: Initialize nonoptimal edges $H := \emptyset$
4: **while** Time is less than t **do**
5: Set $\mathcal{C}_\emptyset{}' := \mathcal{C}_\emptyset$
6: **for all** $F \in \mathcal{C}_\emptyset$ **do**
7: Choose $v \in (V \setminus B) \setminus \mathcal{S}(F)$ (close to c)
8: Let $N := \{x \in V \mid vx \in E\}$
9: Set $\mathcal{F} := \emptyset$
10: **for all** $x, y \in N$ **do**
11: Set $F' := F \cup \{vx, vy\}$
12: Set B' to the border of the induced subinstance of F'
13: **if** LocalOptimality$((V_{F'}, E_{F'}, B', l), F')$ **then**
14: $\mathcal{F} \leftarrow \mathcal{F} \cup \{F'\}$
15: $\mathcal{S}(F') := \mathcal{S}(F) \cup \{v\}$
16: **end if**
17: **end for**
18: $\mathcal{C}_\emptyset{}' \leftarrow (\mathcal{C}_\emptyset{}' \setminus \{F\}) \cup \mathcal{F}$
19: **end for**
20: Set $\mathcal{C}_\emptyset \leftarrow \mathcal{C}_\emptyset{}'$
21: **for all** $e \in E,\ e \nsim \mathcal{C}_\emptyset$ **do**
22: $H \leftarrow H \cup \{e\}$
23: **end for**
24: **end while**
25: **return** H

the outgoing edges are not compatible with the existing set. This reduces the number of sets in the covering. Another criteria is to choose v such that there are very few possibilities of outgoing edge pairs. Note that the chosen vertex v cannot lie in the border B of the subinstance. Otherwise Lemma 3.1 cannot be applied, since some neighbors of v might lie outside of the subinstance. In line 18 the covering is extended. One element is replaced by a set of new sets. It is as well possible, that the set $\mathcal{F}$ is empty. In this case the covering decreases by one element.

This local elimination algorithm is now used in Algorithm 4 working on the whole instance. The implementation of the algorithm allows every calculation of line 7 to be sent to a different process. Hence the calculation can easily be split into $|V_C|$ parallel processes. The subroutine only works on the given subinstance.

Algorithm 4 EdgeElimination

Input: An instance (V, E, l), number of iterations I, timebound t.
Output: A set of nonoptimal edges $H \subset E$.

Set $H := \emptyset$
Select center nodes $V_C \subseteq V$
for $i \leq I$ **do**
    for all $c \in V_C$ **do**
        Set local timebound $t' := \frac{t}{I \cdot |V_C|}$
        Choose V' with $c \in V'$ and induced edge set E'
        $H \leftarrow H \cup$ PathElimination$((V', E' \setminus H, B, l), c, t')$
    end for
end for
return H

5 Results

The proposed algorithm has been tested on all instances from the TSPLIB [8] using the EUC_2D length function. Further some other large instances have been tested. The results are given in Tables 1 and 2. The first two columns contain the name and the dimension of every instance. The next columns refer to the Steps 1 to 3 as described in Section 4. The ratio is the number of remaining edges divided by the number of vertices of the instance after each step. The given times refer to the total runtime for each of the three steps. The last column shows the overall runtime for the algorithm. All times are given in the format `hh:mm:ss` measured on a 2.9GHz Intel Xeon. Due to the parallelization and the inaccuracy of the time measured, the runtimes may be mistaken by up p seconds, when p parallel processes are used. Some values refer to the table given in [6].

Table 1: Results for all EUC_2D TSPLIB instances with more than 1000 vertices, as well as the mona-lisa100K and usa115475 instance from [3].

Instance	n	**Step 1**		**Step 2**		**Step 3**		**Total**
		Time	**Ratio**	**Time**	**Ratio**	**Time**	**Ratio**	**Runtime**
pr1002	1002	1	42.55	2:13	5.80	2:28:07	4.51	2:30:21
u1060	1060	1	41.40	2:24	5.72	3:30:48	4.36	3:33:13
vm1084	1084	1	37.78	3:28	5.57	1:17:35	4.25	1:21:05
pcb1173	1173	1	27.73	33	6.53	3:10:17	5.19	3:10:51
d1291	1291	4	95.20	52:21	9.72	13:33:14	8.77	14:25:40
rl1304	1304	3	95.52	11:54	16.63	12:53:14	11.14	13:05:11
rl1323	1323	2	80.77	5:33	12.66	9:41:18	9.59	9:46:53
nrw1379	1379	1	20.64	1:58	5.22	2:12:44	4.17	2:14:44
fl1400	1400	56	176.49	29:12:48	11.29	2:19:04	10.58	31:32:48
u1432	1432	1	15.34	2:51	5.46	2:13:02	4.54	2:15:55
fl1577	1577	1:20	76.47	3:24:40	30.97	2:42:40	30.04	6:08:40
d1655	1655	9	139.49	37:30	8.67	10:05:46	7.31	10:43:26
vm1748	1748	6	82.77	23:10	7.04	2:48:14	4.40	3:11:30
u1817	1817	5	60.02	10:58	7.27	6:19:22	6.46	6:30:25
rl1889	1889	9	109.46	3:15:40	12.39	25:18:52	9.89	28:34:41
d2103	2103	8	79.35	55:01	9.33	18:19:34	8.61	19:14:44
u2152	2152	5	54.38	11:17	7.02	7:07:45	6.12	7:19:08
u2319	2319	3	9.36	44	4.28	1:41:41	4.08	1:42:22
pr2392	2392	7	50.80	13:03	6.52	7:41:45	5.05	7:44:55
pcb3038	3038	8	31.46	11:05	5.91	5:44:08	4.89	5:55:22
fl3795	3795	8:24	249.20	112:00:00	162.60	0	162.60	112:08:24
fnl4461	4461	15	28.81	9:30	5.37	7:14:21	4.28	7:24:07
rl5915	5915	10:40	57.94	4:40:00	26.99	10:28:00	22.10	15:18:40
rl5934	5934	10:40	54.70	3:24:40	23.08	10:35:20	18.46	14:10:40
rl11849	11849	27:04	42.07	3:46:48	16.96	20:26:24	12.82	24:40:16
usa13509	13509	24:40	24.50	29:20	7.86	26:28:00	4.47	27:22:00
brd14051	14051	4:39	189.44	18:50:15	6.65	28:39:22	4.59	47:34:16
d15112	15112	5:51	112.74	10:25:44	8.61	38:37:42	4.37	49:09:17
d18512	18512	5:30	78.32	1:49:35	6.09	32:38:07	4.55	34:33:13
mona-lisa100K	100000	3:45:21	20.71	22:51	4.76	55:42:51	3.23	59:51:04
usa115475	115475	25:06:00	25.91	76:37:20	9.44	449:38:40	7.15	551:22:00

Table 2: Results for all EUC_2D TSPLIB instances with less than 1000 vertices.

Instance	n	**Step 1**		**Step 2**		**Step 3**		**Total**
		Time	**Ratio**	**Time**	**Ratio**	**Time**	**Ratio**	**Runtime**
eil51	51	<24	18.73	<24	4.49	<24	4.49	<24
berlin52	52	<24	11.83	<24	4.88	<24	4.46	<24
st70	70	<24	26.81	<24	5.79	<24	5.79	<24
eil76	76	<24	28.74	<24	5.29	<24	5.29	<24
pr76	76	<24	12.67	<24	6.03	6:00	3.05	6:00
rat99	99	<24	18.37	<24	5.20	6:40	2.91	6:40
kroA100	100	<24	14.17	<24	5.33	6:40	2.20	6:40
kroB100	100	<24	14.65	40	5.64	6:40	2.45	7:20
kroC100	100	<24	14.68	40	5.27	6:40	1.96	7:20
kroD100	100	<24	15.44	40	5.45	6:40	1.82	7:20
kroE100	100	<24	14.78	<24	5.41	6:40	1.96	6:40
rd100	100	<24	15.81	40	5.22	6:40	2.37	7:20
eil101	101	<24	39.53	<24	5.97	<24	5.97	<24
lin105	105	<24	18.32	<24	6.71	6:40	2.85	6:40
pr107	107	<24	19.35	<24	6.08	6:40	2.57	6:40
pr124	124	<24	24.09	<24	11.41	6:40	1.61	6:40
bier127	127	<24	16.20	<24	5.73	6:40	3.52	6:40
ch130	130	<24	19.49	<24	6.01	6:40	3.28	6:40
pr136	136	<24	14.01	<24	5.82	12:40	2.66	12:40
pr144	144	<24	34.85	1:20	18.44	13:20	3.14	14:40
ch150	150	<24	17.94	40	5.95	13:20	2.30	14:00
kroA150	150	<24	17.07	<24	5.65	12:40	1.99	12:40
kroB150	150	<24	16.39	40	5.90	12:40	2.00	13:20
pr152	152	<24	27.17	40	9.82	19:20	1.55	20:00
u159	159	<24	19.18	<24	7.35	13:20	3.36	13:20
rat195	195	<24	20.41	<24	6.34	13:20	3.64	13:20
d198	198	<24	18.27	<24	6.19	19:20	2.70	19:20
kroA200	200	<24	18.08	<24	6.09	19:20	2.15	19:20
kroB200	200	<24	17.70	<24	6.25	18:40	2.04	18:40
ts225	225	<24	20.30	40	13.46	19:20	5.94	20:00
tsp225	225	<24	22.50	40	5.69	19:20	2.45	20:00
pr226	226	40	53.65	5:20	29.35	20:40	23.05	26:40
gil262	262	40	52.31	40	7.97	20:00	4.29	21:20
pr264	264	<24	38.16	2:40	10.89	28:00	5.33	30:40
a280	280	<24	29.01	40	8.47	26:40	4.75	27:20
pr299	299	<24	17.34	40	7.10	26:40	2.62	27:20
lin318	318	<24	26.39	40	8.58	26:40	3.36	27:20
rd400	400	<24	22.54	1:20	6.54	48:40	3.04	50:00
fl417	417	<24	40.51	2:48	6.35	48:32	3.83	51:20
pr439	439	<24	23.92	40	9.26	44:00	3.86	44:40
pcb442	442	<24	18.98	40	7.74	40:00	4.36	40:40
d493	493	40	21.32	40	6.38	46:40	3.39	48:00
u574	574	<24	23.26	1:20	7.20	58:00	3.14	59:20
rat575	575	<24	28.93	40	6.53	54:00	3.77	54:40
p654	654	<24	31.10	3:20	6.16	1:33:20	4.26	1:36:40
d657	657	<24	22.73	40	7.48	1:04:40	3.65	1:05:20
u724	724	<24	20.33	1:20	6.60	1:12:40	2.89	1:14:00
rat783	783	<24	37.49	1:20	7.27	1:23:20	4.25	1:24:40

In Step 3 of the proposed algorithm optimal path systems are calculated. The algorithm starts with all possible edge pairs leaving a specific vertex. We call this a covering of depth 1. The algorithm extends this covering by replacing every element by a covering of this element. After this extension we call the obtained path system a covering of depth 2 and so on. Figure 8 shows the average number of elements in an optimal path system of a given depth. Every line corresponds to a path system which has been extended up to the depth given by the label of the line. For a line with label z, the position (x, y) can be interpreted as follows. Given an optimal path system which has been extended up to depth z. Replace every element of the path system by the subset corresponding to an extension of depth x only. Obviously the remaining set still is an optimal path system. Then on average y is the number of elements of this covering. We can for example consider the optimal path systems with extension depth 2. On average there are about 20 elements in such a path system, when we only extend the path system to depth 2 (shown by the blue line). The number of elements of this path system reduces to around 10, if the calculation is extended up to depth 16 (shown by the yellow line).

The values of Figure 8 are the average values taken from all covering calculations done on the mona-lisa100K instance.

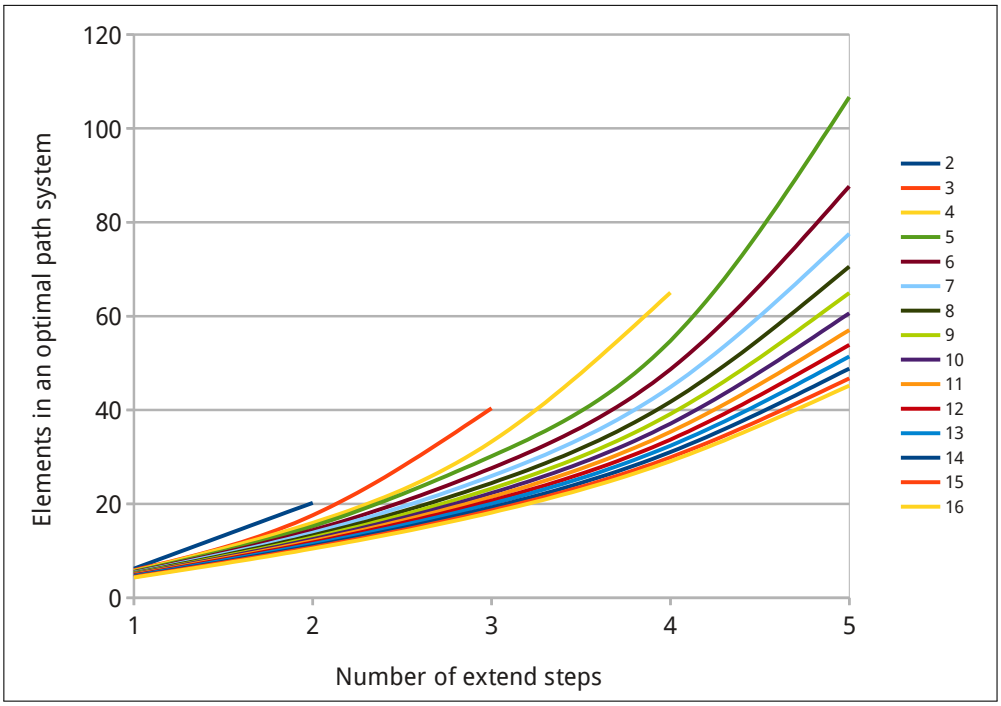

Figure 8: The average number of elements in an optimal path system.

Figure 9 shows the set of edges left after applying the proposed algorithm.

Figure 9: An 263,047 edge set of the mona-lisa100K instance containing all optimal tours. The blue edges are proven to be optimal.

References

[1] D.L.Applegate et al. Op. Res. Let. 37, 11-15 2009

[2] D.L.Applegate, R.E.Bixby, V.Chvátal, W.J.Cook. The Traveling Salesman Problem. A Computational Study. Princeton University Press, 2006

[3] W.J.Cook's TSP website at `http://www.math.uwaterloo.ca/tsp/`

[4] G.Dantzig, R.Fulkerson, S.Johnson. Solution of a large-scale traveling-salesman problem. Journal of the Operations Research Society of America, Vol. 2, No. 4 (Nov., 1954), pp. 393–410

[5] M.R.Garey, D.S.Johnson. Computers and Intractability. Freeman, New York 1979

[6] S.Hougardy, R.Schroeder. Edge Elimination in TSP Instances. arXiv:1402.7301v1, 2014

[7] R.Jonker, T.Volgenant. Nonoptimal edges for the symmetric traveling salesman problem. Operations Research, Vol. 32, No. 4 (1984), pp. 837–846

[8] G.Reinelt. TSPLIB 95. Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Heidelberg, 1995