Formalizing Theory of Approximating the Traveling-Salesman Problem

Lukas Koller

May 26, 2023

Abstract

The Traveling Salesman Problem (TSP) is a well-known NP-complete optimization problem. The Metric TSP is a subcase of the TSP which is still NP-hard. In this report, I present my formalization and formal verification of two approximation algorithms for the Metric TSP: the DoubleTree and Christofides-Serdyukov algorithm. I also present a formalization of a linear reduction (L-reduction) from the Minimum Vertex Cover Problem, with maximum degree of 4 (VCP4), to the Metric TSP. The L-reduction proves the MaxSNP-hardness of the Metric TSP.

Contents

1 Introduction

The Traveling-Salesman Problem (TSP) is a well-known optimization problem that describes the task of finding a tour among the vertices of a given weighted and undirected graph s.t. the tour has minimum total weight and every vertex of the graph is visited exactly once. The TSP has many applications in different areas, such as planning and scheduling [3], logistics [14], and designing printed circuit boards [7].

The TSP is NP-hard [9, Theorem 15.43]. This means, that unless P = NP, the TSP cannot be solved in polynomial time. When dealing with NP-hard optimization problems, a typical strategy is to approximate the optimal solution with an approximation algorithm. An approximation algorithm [16] is a polynomial time algorithm that returns a bounded approximate solution. An approximation algorithm with an approximation ratio of α is guaranteed to return an approximate solution that in the worst case has a total cost of αOPT, where OPT is the total cost of the optimal solution. Ideally, the approximation ratio is a constant factor. An approximation algorithm essentially gives up the optimality of the returned solution for a polynomial running time. Unfortunately, there is bad news for the TSP: there is no constant-factor approximation algorithm for the TSP [9, Theorem 21.1].

Thus, in this work, I only consider the Metric TSP. The Metric TSP is a subcase of the TSP which makes the following two assumptions: (i) the given graph is complete and (ii) the edge weights satisfy the triangle inequality. The Metric TSP is still NP-hard [9, Theorem 21.2].

In this work, present my formalization of parts of section 21.1, *Approximation Algorithms for the TSP*, from [9].

1. I formalize and formally verify two approximation algorithms, DoubleTree and Christofides-Serdyukov, for Metric TSP.

2. I formalize an L-reduction from the Minimum Vertex Cover Problem with maximum degree of 4 (VCP4) to the Metric TSP, which proves the MaxSNP-hardness of the Metric TSP.

Unless P = NP, the MaxSNP-hardness of a problem proves the non-existence of an approximation scheme [9, Corollary 16.40]. An approximation scheme takes a parameter $\varepsilon > 0$ and produces an approximation algo-

rithm with an approximation ratio of $(1 + \varepsilon)$ for minimization and $(1 - \varepsilon)$ for maximization problems. MaxSNP is a complexity class that contains graph-theoretical optimization problems. For any problem in MaxSNP there is a constant-factor approximation algorithm [12, Theorem 1]. A linear reduction (L-reduction) [12] is a special type of reduction for optimization problems, that is used to prove the MaxSNP-hardness of problems. A problem is called MaxSNP-hard if every problem in MaxSNP L-reduces to it. The VCP4 is known to be MaxSNP-hard [9, Theorem 16.46].

All of my formalizations are done with the interactive theorem prover Isabelle/HOL [11]. My formalizations can be found on GitHub: https://github.com/kollerlukas/tsp.

2 Related Work

The Christofides-Serdyukov algorithm [4, 15] has an approximation ratio of $\frac{3}{2}$, and thus is the best known approximation algorithm for the Metric TSP [9]. Recently, a randomized approximation algorithm for the Metric TSP was proposed, which (in expectation) achieves a slightly better approximation ratio than the Christofides-Serdyukov algorithm [8].

In [6], different approximation algorithms are formalized and verified in Isabelle/HOL. I formalize the DoubleTree and the Christofides-Serdyukov algorithm in similar fashion by giving an implementation for both of the approximation algorithms in a simple imperative `WHILE`-language.

In [17] polynomial time reductions are formalized in Isabelle/HOL. To formally reason about the running times of functions, a computational model for the programming language is required. Therefore, [17] have developed the imperative language `IMP-` which provides a computational model to reason about the running times of functions implemented in `IMP-`. To my knowledge, there are no previous attempts to formalize an L-reduction in Isabelle/HOL.

3 Fundamentals and Definitions

I build on the formalization of graphs by [1], which is developed for a formalization of Berge's theorem [2]. An undirected graph is represented by a finite set of edges. An edge is represented by a doubleton set.

$$\textit{graph-invar} \equiv \lambda E.\ (\forall\, e{\in}E.\ \exists\, u\ v.\ e = \{u, v\} \wedge u \neq v) \wedge \textit{finite}\ (\textit{Vs}\ E)$$

The locale *graph-abs* fixes an instance of a graph E that satisfies the invariant *graph-invar*. The graph formalization by [1] provides definitions for the following concepts.

- *Vs* returns the set of vertices of a graph.
- A path is represented with a list of vertices and is characterized by the predicate *path*. A walk (*walk-betw*) is a path between two specific vertices. The function *edges-of-path* returns the list of edges of a given path.
- *degree* returns the degree of a vertex in a graph.
- *connected-component* return the connected component of a vertex in a graph.
- A matching is a graph where no two edges share an endpoint. The predicate *matching* characterizes matchings.

The approximation algorithms and the L-reduction use many graph-related concepts that are not defined in [1], such as weighted graphs, minimum spanning trees, or Hamiltonian cycles. This section describes the formalization of necessary graph-related concepts. Most of the definitions are straightforward, therefore only important formalization choices are highlighted.

All of the graph-related definitions are defined in theories that are located in the `./graphs-` or `./problems`-directory. Many simple results and lemmas are formalized in the theory `./misc/Misc.thy` which is located in the directory `./misc`.

3.1 Connected Graphs

A graph E is connected if all vertices are contained in the same connected component.

is-connected $E \equiv \forall\, u{\in}\mathit{Vs}\ E.\ \forall\, v{\in}\mathit{Vs}\ E.\ u \in \mathit{connected\text{-}component}\ E\ v$

3.2 Weighted Graphs

The locale *w-graph-abs* fixes an instance of a graph E and edge weights c. The locale *pos-w-graph-abs* extends the locale *w-graph-abs* with the assumption that all edge weights are positive.

The function $\mathit{cost\text{-}of\text{-}path}_c$ computes the cost of a given path with the edge weights c.

3.3 Complete Graphs

A graph E is complete if there exists an edge between every two vertices.

$is\text{-}complete\ E \equiv \forall\, u\ v.\ u \in Vs\ E \wedge v \in Vs\ E \wedge u \neq v \longrightarrow \{u, v\} \in E$

The locale *compl-graph-abs* fixes an instance of a complete graph E.

In a complete graph E, any sequence of vertices P s.t. no two consecutive vertices are equal (*distinct-adj*) is a path.

$is\text{-}complete\ E \wedge distinct\text{-}adj\ P \wedge P \subseteq Vs\ E \Longrightarrow path\ E\ P$

The locale *metric-graph-abs* extends the locales *compl-graph-abs* and *pos-w-graph-abs* and assumes that the edge weights c satisfy the triangle inequality. Within such a graph any intermediate vertex on a path can be removed to obtain a shorter path. Given a set of vertices X and a path P, the function *short-cut* removes all vertices along the path P that are not contained X. The following lemma proves that the resulting path has less total weight.

$P \subseteq Vs\ E \Longrightarrow cost\text{-}of\text{-}path_c\ (short\text{-}cut\ X\ P) \leq cost\text{-}of\text{-}path_c\ P$

3.4 Acyclic Graphs

A path P is simple if the path uses every edge at most once.

$is\text{-}simple\ P \equiv distinct\ (edges\text{-}of\text{-}path\ P)$

A cycle C is a walk s.t. (i) the first and last vertex are the same, (ii) the path is simple, and (iii) the walk uses at least one edge.

$is\text{-}cycle\ E\ C \equiv (\exists\, v.\ walk\text{-}betw\ E\ v\ C\ v) \wedge is\text{-}simple\ C \wedge 0 < |edges\text{-}of\text{-}path\ C|$

A graph E is acyclic if it does not contain a cycle.

$is\text{-}acyclic\ E \equiv \nexists\, C.\ is\text{-}cycle\ E\ C$

The following lemma proves the fact that a vertex v that is contained in a cycle C has a degree of at least 2.

$is\text{-}cycle\ E\ C \wedge v \in C \Longrightarrow 2 \leq degree\ E\ v$

3.5 Trees

A tree T is a graph that is (i) connected, and (ii) acyclic.

$$\textit{is-tree}\ T \equiv \textit{is-connected}\ T \wedge \textit{is-acyclic}\ T$$

A spanning tree T of a graph E is a subgraph s.t. (i) T is a tree and (ii) T contains every vertex of E.

$$\textit{is-st}\ E\ T \equiv T \subseteq E \wedge \textit{Vs}\ E = \textit{Vs}\ T \wedge \textit{is-tree}\ T$$

A minimum spanning tree (MST) T is a spanning tree with minimum total weight.

$$\textit{is-mst}\ E\ c\ T \equiv \textit{is-st}\ E\ T \wedge (\forall\, T'.\ \textit{is-st}\ E\ T' \longrightarrow (\sum e{\in}T.\ c\ e) \leq (\sum e{\in}T'.\ c\ e))$$

The locale *mst* assumes the existence of an algorithm (denoted by *comp-mst*) to that computes a MST for a given graph.

3.6 Hamiltonian Cycles

A Hamiltonian cycle H is a tour in a graph E that visits every vertex exactly once.

$$\begin{aligned}&\textit{is-hc}\ E\ H \equiv\\ &\quad (H \neq [] \longrightarrow (\exists\, v.\ \textit{walk-betw}\ E\ v\ H\ v)) \wedge \textit{set}\ (\textit{tl}\ H) = \textit{Vs}\ E \wedge \textit{distinct}\ (\textit{tl}\ H)\end{aligned}$$

With this formalization, a Hamiltonian cycle is not necessarily a cycle (*is-cycle*). The graph that only contains one edge $e{=}\{u,v\}$ has the Hamiltonian cycle u,v,u which is not a cycle. The Hamiltonian cycle is not simple, because the edge e is used twice.

3.7 Traveling-Salesman Problem

A solution H to the TSP is a Hamiltonian cycle with minimum total weight.

$$\begin{aligned}&\textit{is-tsp}\ E\ c\ H \equiv \textit{is-hc}\ E\ H\\ &\quad \wedge (\forall\, H'.\ \textit{is-hc}\ E\ H' \longrightarrow \textit{cost-of-path}_c\ H \leq \textit{cost-of-path}_c\ H')\end{aligned}$$

3.8 Multi-Graphs

A multigraph is formalized as a multiset of edges. The theory `./graphs/MultiGraph.thy` contains unfinished attempts for encoding a multigraph with a simple graph.

3.9 Eulerian Tours

A graph E is Eulerian if every vertex has even degree. The DoubleTree and the Christofides-Serdyukov algorithm use Eulerian multigraphs, thus the predicate *is-eulerian* is defined for multigraphs. The function *mdegree* returns the degree of a vertex in a multigraph. *mVs* returns the set of vertices of a multigraph.

$$\textit{is-eulerian}\ E \equiv \forall\, v{\in}\textit{mVs}\ E.\ \textit{even}'\ (\textit{mdegree}\ E\ v)$$

An Eulerian tour P is a path that uses every edge of a given graph E exactly once. The predicate *mpath* characterizes a path in a multigraph.

$$\textit{is-et}\ E\ P \equiv \textit{mpath}\ E\ P \wedge \textit{hd}\ P = \textit{last}\ P \wedge E = \textit{mset}\ (\textit{edges-of-path}\ P)$$

The locale *eulerian* fixes an algorithm (denoted by *comp-et*) that computes an Eulerian tour for a given multigraph.

3.10 Perfect Matchings

A subgraph M is a perfect matching for a graph E if M contains every vertex of E.

$$\textit{is-perf-match}\ E\ M \equiv M \subseteq E \wedge \textit{matching}\ M \wedge \textit{Vs}\ M = \textit{Vs}\ E$$

A minimum perfect matching M is a perfect matching with minimum total weight.

$$\begin{aligned}&\textit{is-min-match}\ E\ c\ M \equiv \textit{is-perf-match}\ E\ M\\ &\quad \wedge\ (\forall\, M'.\ \textit{is-perf-match}\ E\ M' \longrightarrow (\textstyle\sum e{\in}M.\ c\ e) \leq (\sum e{\in}M'.\ c\ e))\end{aligned}$$

The locale *min-weight-matching* fixes an algorithm (denoted by *comp-match*) that computes a minimum perfect matching for a given graph.

3.11 Vertex Cover

A vertex cover X is a subset of the vertices of a graph E s.t. X contains at least one endpoint of every edge of E.

$$\textit{is-vc}\ E\ X \equiv (\forall\, e{\in}E.\ e \cap X \neq \emptyset) \wedge X \subseteq \textit{Vs}\ E$$

A minimum vertex cover is a vertex cover with minimum cardinality.

$$\textit{is-min-vc}\ E\ X \equiv \textit{is-vc}\ E\ X \wedge (\forall\, X'.\ \textit{is-vc}\ E\ X' \longrightarrow |X| \leq |X'|)$$

If the degree of every vertex in a graph E is bounded by a constant k, then the number of edges of E is upper bounded by k-times the cardinality of any vertex cover X for E.

$(\forall\, v{\in}\mathit{Vs}\ E.\ \mathit{degree}\ E\ v \leq \mathit{enat}\ k) \wedge \mathit{is\text{-}vc}\ E\ X \Longrightarrow |E| \leq k * |X|$

4 Approximating the Metric TSP

This section describes the formalization of the DoubleTree and the Christofides-Serdyukov approximation algorithms for the Metric TSP. Both algorithms are quite similar and depend on the following proposition [9, Lemma 21.3]:

Let the graph E with edge weights c be an instance of the Metric TSP. Given a connected Eulerian graph E' that spans the vertices of the graph E, we can compute (in polynomial time) a Hamiltonian cycle for E with a total weight of at most the total weight of all edges in E'.

The proof for this proposition is constructive: first compute an Eulerian tour P for the Eulerian graph E'. The tour P visits every vertex of E at least once, because the Eulerian graph E' spans the vertices of E. Next, the duplicate vertices in the tour P are removed to obtain the desired Hamiltonian cycle for the graph E ("shortcutting"). By assumption, The edge weights c satisfy the triangle inequality, thus the total weight of the resulting Hamiltonian cycle is at most the total weight of the Eulerian tour P, which is exactly the total weight of all edges in the Eulerian graph E'.

The construction for this proposition is formalized with the function *comp-hc-of-et*. Given a Eulerian tour for E', the function *comp-hc-of-et* computes a Hamiltonian cycle for E. The second argument to the function *comp-hc-of-et* is an accumulator.

We assume the multigraph E' spans the vertices of the graph E and only contains edges (maybe multiple instances of an edge) of the graph E. Then, the function *comp-hc-of-et* computes a Hamiltonian cycle for E.

$\mathit{is\text{-}et}\ E'\ P \wedge \mathit{mVs}\ E' = \mathit{Vs}\ E \wedge \mathit{set\text{-}mset}\ E' \subseteq E \Longrightarrow \mathit{is\text{-}hc}\ E\ (\mathit{comp\text{-}hc\text{-}of\text{-}et}\ P\ [])$

The following lemma shows that the total weight of the Hamiltonian cycle returned by the function *comp-hc-of-et* is bounded by the total weight of the Eulerian tour P.

$P \subseteq \mathit{Vs}\ E \Longrightarrow \mathit{cost\text{-}of\text{-}path}_c\ (\mathit{comp\text{-}hc\text{-}of\text{-}et}\ P\ []) \leq \mathit{cost\text{-}of\text{-}path}_c\ P$

With the proposition [9, Lemma 21.3], the task of approximating Metric TSP boils down to constructing an Eulerian graph that spans the vertices

of the graph E. The total weight of the edges of the Eulerian graph directly bounds the total weight of the approximate solution, and thus directly affects the approximation-ratio.

Both approximation algorithms, DoubleTree and Christofides-Serdyukov, first compute a MST T for the graph E. Then, edges are added to the MST T to construct a Eulerian multigraph E' that spans the vertices of the graph E. The construction from [9, Lemma 21.3] is then applied to E' to obtain a Hamiltonian cycle for E.

The DoubleTree algorithm constructs an Eulerian graph by simply doubling every edge of the MST T. The total weight of any Hamiltonian cycle is at least the total weight of T. Thus, the total weight of the Hamiltonian cycle produced by the DoubleTree algorithm is at most double the total weight of the optimal Hamiltonian cycle. Therefore, the DoubleTree algorithm is a 2-approximation algorithm for the Metric TSP.

On the other hand, the Christofides-Serdyukov algorithm computes a minimum perfect matching M between the vertices that have odd degree in T. The union of the matching M and the MST T is a Eulerian multigraph. The total weight of M is at most half the total weight of the optimal Hamiltonian cycle. Therefore, the Christofides-Serdyukov algorithm is a $\frac{3}{2}$-approximation algorithm for the Metric TSP.

The polynomial running time of both the DoubleTree and Christofides-Serdyukov are easy to see and thus explicit proofs are omitted.

4.1 Formalizing the DoubleTree Algorithm

The DoubleTree algorithm consists of three steps.

1. Compute a MST T of the input graph E.
2. Compute an Eulerian tour P of the doubled MST $T + T$.
3. Remove duplicate vertices in P by short-cutting (see *comp-hc-of-et*).

Thus, the DoubleTree algorithm depends on two algorithms:

1. an algorithm to compute a MST, e.g. Prim's algorithm [10], and
2. an algorithm to compute a Eulerian tour in an Eulerian multigraph, e.g. Eulers's algorithm [9].

For the formalization of the DoubleTree algorithm the existence of an algorithm for both of these problems is assumed. The function *comp-mst* denotes the assumed algorithm to compute a MST, and the function *comp-et* denotes the assumed algorithm to compute an Eulerian tour.

```
double-tree = (
    let T = comp-mst c E;
        T_2x = mset-set T + mset-set T;
        P = comp-et T_2x in
          comp-hc-of-et P [])
```

For the defined DoubleTree algorithm two properties are proven: (i) the feasibility, and (ii) the approximation ratio. The formalization of these proofs is straightforward. The following lemma proves the feasibility of the DoubleTree algorithm. The path returned by the DoubleTree algorithm is indeed a Hamiltonian cycle for the graph E.

is-hc E double-tree

The following lemma shows that the approximation ratio of the DoubleTree algorithm is 2. OPT denotes the optimal tour for the graph E.

$cost\text{-}of\text{-}path_c\ double\text{-}tree \leq 2 * cost\text{-}of\text{-}path_c\ OPT$

Finally, the definition of the DoubleTree algorithm is refined to a `WHILE`-program using Hoare Logic.

```
{True}
T := comp-mst c E;
T_2 := mset-set T + mset-set T;
P := comp-et T_2;
P' := P;
H := [];
WHILE P' ≠ []
INV {comp-hc-of-et P [] = comp-hc-of-et P' H ∧
       P = comp-et T_2 ∧ T_2 = mset-set T + mset-set T ∧ T = comp-mst c E}
VAR {0}
DO v := hd P';
    P' := tl P'; IF v ∈ H ∧ P' ≠ [] THEN SKIP  ELSE H := v · H FI
OD
{is-hc E H ∧ cost-of-path_c H ≤ 2 * cost-of-path_c OPT}
```

4.2 Formalizing the Christofides-Serdyukov Algorithm

The Christofides-Serdyukov algorithm is similar to the DoubleTree, and consists of the following steps.

1. Compute a MST T of the input graph E.
2. Compute a minimum perfect matching M between the vertices that have odd degree in T.

3. Compute a Eulerian tour P of the union of the MST and the perfect matching $J = T + M$.

4. Remove duplicate vertices in P by short-cutting (see *comp-hc-of-et*).

Hence, the Christofides-Serdyukov algorithm depends on three algorithms: the algorithms the DoubleTree algorithm already depends on, as well as an algorithm to compute a minimum perfect matching, e.g. Edmond's Blossom algorithm [5].

The Christofides-Serdyukov algorithm is formalized similarly to the DoubleTree algorithm. The function *comp-match* denotes an assumed algorithm that computes a minimum perfect matching for a given graph.

```
christofides-serdyukov = (
  let T = comp-mst c E;
      W = {v ∈ Vs T. ¬ even' (degree T v)};
      M = comp-match ({e ∈ E. e ⊆ W}) c;
      J = mset-set T + mset-set M;
      P = comp-et J in
    comp-hc-of-et P [])
```

The feasibility of the Christofides-Serdyukov algorithm is proven by the following lemma. The path returned by the Christofides-Serdyukov algorithm is indeed a Hamiltonian cycle.

is-hc E christofides-serdyukov

The following lemma shows that the approximation ratio of the Christofides-Serdyukov algorithm is $\frac{3}{2}$. OPT denotes the optimal tour for the graph E.

$2 * \textit{cost-of-path}_c\ \textit{christofides-serdyukov} \leq 3 * \textit{cost-of-path}_c\ OPT$

Like the DoubleTree algorithm, the definition of the Christofides-Serdyukov algorithm is refined to a `WHILE`-program using Hoare Logic.

```
{True}
T := comp-mst c E;
W := {v ∈ Vs T | ¬ even' (degree T v)};
M := comp-match {e ∈ E | e ⊆ W} c;
J := mset-set T + mset-set M;
P := comp-et J;
P' := P;
H := [];
WHILE P' ≠ []
```

$$
\begin{array}{l}
\mathit{INV}\ \{\mathit{comp\text{-}hc\text{-}of\text{-}et}\ P\ [] = \mathit{comp\text{-}hc\text{-}of\text{-}et}\ P'\ H\ \wedge \\
\quad\quad P = \mathit{comp\text{-}et}\ J\ \wedge \\
\quad\quad J = \mathit{mset\text{-}set}\ T + \mathit{mset\text{-}set}\ M\ \wedge \\
\quad\quad M = \mathit{comp\text{-}match}\ \{e \in E \mid e \subseteq W\}\ c\ \wedge \\
\quad\quad W = \{v \in \mathit{Vs}\ T \mid \neg\ \mathit{even}'\ (\mathit{degree}\ T\ v)\} \wedge T = \mathit{comp\text{-}mst}\ c\ E\} \\
\mathit{VAR}\ \{0\} \\
\mathit{DO}\ v := \mathit{hd}\ P'; \\
\quad P' := \mathit{tl}\ P';\ \mathit{IF}\ v \in H \wedge P' \neq []\ \mathit{THEN\ SKIP}\ \ \mathit{ELSE}\ H := v \cdot H\ \mathit{FI} \\
\mathit{OD} \\
\{\mathit{is\text{-}hc}\ E\ H \wedge 2 * \mathit{cost\text{-}of\text{-}path}_c\ H \leq 3 * \mathit{cost\text{-}of\text{-}path}_c\ \mathit{OPT}\}
\end{array}
$$

5 L-Reduction from VCP4 to Metric TSP

This section describes the formalization of an L-Reduction from the Minimum Vertex Cover Problem, with maximum degree of 4 (VCP4), to the Metric TSP. The formalization is based on an L-reduction proof by [9, Theorem 21.1].

The Minimum Vertex Cover Problem describes the optimization problem of finding a vertex cover with minimum cardinality for a given graph. The VCP4 is the restriction of the Minimum Vertex Cover Problem to input graphs where the degree of every vertex is bounded by 4. The bounded degree is only needed to prove the linear inequalities that are required for the L-reduction. The VCP4 is known to be MaxSNP-hard [9, Theorem 16.46].

First, I define what an L-reduction is. Let A and B be optimization problems with cost functions c_A and c_B. The cost of the optimal solution for an instance of an optimization problem is denoted by $OPT(\cdot)$. An L-reduction (linear reduction) consists of a pair of functions f and g (both computable in polynomial time) and two positive constants α,$\beta > 0$ s.t. for every instance x_A of A

(i) $f\ x_A$ is an instance of B s.t.
$OPT\ (f\ x_A) \leq \alpha(OPT\ x_A)$, and

(ii) and for any feasible solution y_B of $f\ x_A$, $g\ x_A\ y_B$ is a feasible solution of x_A s.t.
$|(c_A\ x_A\ (g\ x_A\ y_B)) - OPT\ x_A| \leq |(c_A\ (f\ x_A)\ y_B) - OPT\ (f\ x_A)|$.

The second linear inequality essentially ensures the following: given an optimal solution for $f\ x_A$ the function g has to construct an optimal solution for x_A.

The VCP4 is L-reduced to the Metric TSP by defining the functions f and g and proving the feasibility of the functions and the required inequalities.

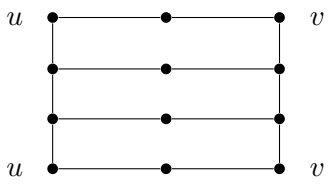
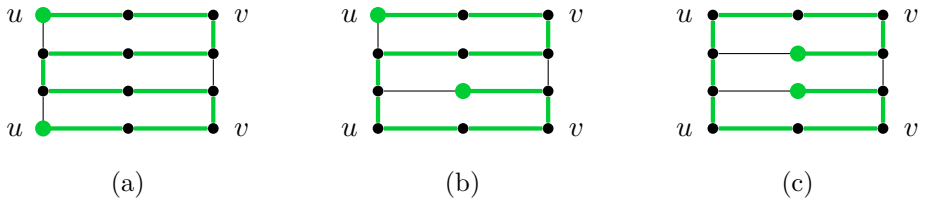


Figure 1: Subgraph H_e for an edge $e{:=}\{u,v\}$. Each corner vertex of H_e corresponds to an endpoint of e.



(a) (b) (c)

Figure 2: The different types of Hamiltonian paths for the subgraphs H_e for an edge $e{:=}\{u,v\}$.

The function f maps an instance of the VCP4 to an instance of the Metric TSP. An instance of the VCP4 consists of a graph where the degree of every vertex is at most 4. To construct an instance of the Metric TSP, we need to construct a complete graph with edge weights s.t. the edge weights satisfy the triangle inequality.

The function f is defined by the following construction. Let G be an instance of the VCP4, and let $H{:=}f\ G$ denote the graph that is constructed from G by the function f. For each edge e of G, a subgraph H_e is added to H (see Figure 1). The function f computes the complete graph over all the subgraphs H_e (one for every edge e of the graph G). A subgraph H_e consists of 12 vertices that are arranged in a 4-by-3 lattice. Each corner vertex of a subgraph H_e corresponds to an endpoint of the edge e. Moreover, a subgraph H_e has the special property that there is no Hamiltonian path for H_e that starts and ends at corner vertices of H_e that correspond to different endpoints of the edge e. E.g. there is no Hamiltonian path for the subgraph H_e that starts at the top-left corner vertex and ends at the bottom-right corner vertex. Therefore, the start- and end-vertex of a Hamiltonian path for a subgraph H_e can only correspond to the one endpoint of the edge e. This property is used to encode a vertex cover of G with a Hamiltonian cycle of H. The subgraph H_e admits 3 types of Hamiltonian paths (see Figure 2).

Next, I describe the edge weights of the graph H. The graph H is complete, thus there is an edge between every pair of vertices of H. Every vertex of H belongs to exactly one subgraph H_e. We distinguish between 3 types of edges.

(i) An edge that connects two vertices that both belong to the same subgraph H_e has a weight equal to the distance of the vertices in the subgraph H_e.

(ii) An edge that connects two corner vertices of different subgraphs H_e and H_f ($e \neq f$) but both of the vertices correspond to the same endpoint v in G has a weight of 4.

(iii) All remaining edges have a weight of 5.

The edge weights satisfy the triangle inequality because the edge weights can be seen as a metric completion of the graph H restricted to the edges that have weight 1 or 4.

Next, I describe the definition of the function g. The function g maps a Hamiltonian cycle T for H to a vertex cover X of G. By the construction of H, the Hamiltonian cycle T may be composed of only Hamiltonian paths for the subgraphs H_e. In this case, for each edge e of G the covering vertex of e is identified by looking at the Hamiltonian path for the subgraph H_e that is contained in T. The Hamiltonian path of the subgraph H_e can only correspond to one endpoint of the edge e. This endpoint is selected as the covering vertex for the edge e. If the Hamiltonian cycle T does not contain a Hamiltonian path for every subgraph H_e, a Hamiltonian cycle T' for H is constructed. The Hamiltonian cycle T' contains a Hamiltonian path for every subgraph H_e and the total cost of T' is at most the total cost of T. The Hamiltonian cycle T' is constructed by carefully replacing parts of T with a Hamiltonian path for a subgraph H_e.

5.1 Formalizing the L-Reduction

The locale *ugraph-adj-map* provides an abstract adjacency map that serves as a graph representation that allows for the implementation of executable functions on graphs. The locale *ugraph-adj-map* can be instantiated with an implementation for sets (locale *Set2*) and maps (locale *Map*) to obtain executable functions on graphs.

The set of undirected edges of a graph is returned by the function $E(\cdot)$. A linear order on the vertices is used to identify different instances of the same undirected edge.

The L-reduction itself is formalized in the locale *VCP4-To-mTSP* that depends on two executable adjacency-maps (locale *ugraph-adj-map*), one for each of the graphs G and H which are denoted by *g1* and *g2*. The locale *VCP4-To-mTSP* also assumes functions that provide `fold`-operations for the graphs. The reduction functions f and

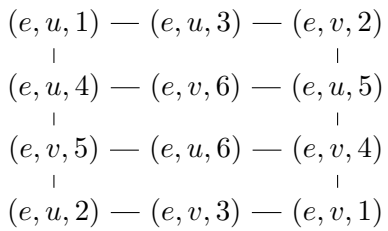


Figure 3: Formalization of the subgraph H_e for an edge e:=$\{u,v\}$.

g are defined in the locale *VCP4-To-mTSP* using the assumed `fold`-operations and some auxiliary functions. The theory `reductions/VertexCover4ToMetricTravelingSalesman_AdjList.thy` instantiates the locale *VCP4-To-mTSP* with an implementation of an adjacency list to obtain executable functions.

The vertices of the subgraphs H_e are represented by a tuple (e,w,i) where w is an endpoint of the edge e and $i \in \{1..6\}$ is an index. Figure 3 shows the formalization of a subgraph H_e. The vertices of the subgraph H_e have to be named such that the subgraph is symmetric, i.e. $H_{\{u,v\}}$=$H_{\{v,u\}}$.

The function f uses the auxiliary function *complete-graph* which constructs the complete graph for a given set of vertices. The function V_H computes all the vertices of the graph H.

$f\ G = \textit{complete-graph}\ (V_H\ G)$

The edge weights are formalized by the function c, which maps two vertices to an integer. The definition of the function c uses a couple of auxiliary functions to distinguish different cases. To simplify things, the case for two vertices that are neighbors in subgraph H_e (weight 1) is handled separately. The function *is-edge-in-He* checks if there is a subgraph H_e in H where the two given vertices are connected by an edge. The function *are-vertices-in-He* checks if there is a subgraph H_e in H that contains both given vertices. The function *min-dist-in-He* computes the distance between two vertices in a subgraph H_e. The function *rep1* is used to identify different instances of the undirected edge.

$c\ G\ (e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2) =$
$(\textsf{if}\ \textit{is-edge-in-He}\ G\ (\textit{uEdge}\ (e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2))\ \textsf{then}\ 1$
$\ \textsf{else if}\ \textit{are-vertices-in-He}\ G\ (e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2)$
$\quad \textsf{then}\ \textit{the-enat}\ (\textit{min-dist-in-He}\ G\ (e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2))$
$\quad \textsf{else if}\ \textit{rep1}\ e_1 \neq \textit{rep1}\ e_2 \wedge$
$\qquad w_1 = w_2 \wedge (i_1 = 1 \vee i_1 = 2) \wedge (i_2 = 1 \vee i_2 = 2)$
$\qquad \textsf{then}\ 4\ \textsf{else}\ 5)$

The function g depends on two functions *shorten-tour* and *vc-of-tour*. The function *shorten-tour* modifies a Hamiltonian cycle s.t. the resulting tour

has less total weight and the tour contains a Hamiltonian path for every subgraph H_e. The function *vc-of-tour* computes a vertex cover for G given a Hamiltonian cycle that contains a Hamiltonian path for every subgraph H_e.

g G T = vc-of-tour G (tl (shorten-tour G T))

Given a tour T of H, the function *shorten-tour* iteratively (for each edge e of G) removes all vertices of the subgraph H_e from the tour T and inserts a Hamiltonian path for the subgraph H_e. The Hamiltonian path is inserted at the position where the tour T enters the subgraph H_e for the first time. This replacement of the Hamiltonian path is done regardless of whether the tour T already contains a Hamiltonian path for the subgraph H_e. This simplifies definitions and proofs by avoiding extra case distinctions. The inserted Hamiltonian path for the subgraph H_e is carefully chosen such that the resulting tour has less total weight. Let $e{:=}\{u,v\}$, and see Figure 3 for the naming of the vertices in the subgraph H_e.

1. If the tour T does not contain a Hamiltonian path for the subgraph H_e then the Hamiltonian path that starts at $(e,u,1)$ and ends at $(e,u,2)$ is inserted.

2. If the Hamiltonian path for H_e in the tour T starts at a corner vertex (e,w,i), with $i{\in}\{1,2\}$ and $w{\in}\{u,v\}$, then the Hamiltonian path in the tour T is replaced with the Hamiltonian that starts at (e,w,i) and ends at (e,w,j), where $j{\in}\{1,2\}{-}\{i\}$.

3. If the Hamiltonian path for H_e in the tour T ends at a corner vertex (e,w,i), with $i{\in}\{1,2\}$ and $w{\in}\{u,v\}$, then the Hamiltonian path in the tour T is replaced with the Hamiltonian that starts at (e,w,j) and ends at (e,w,i), where $j{\in}\{1,2\}{-}\{i\}$.

4. Otherwise, the Hamiltonian path that starts at $(e,u,1)$ and ends at $(e,u,2)$ is inserted.

The function *shorten-tour* formalizes property (b) from the L-reduction proof in [9, Theorem 21.1].

5.2 Proofs for the L-Reduction

The locale *VCP4-To-mTSP* contains the proofs for the feasibility of the reduction functions and the linear inequalities required for an L-reduction.

The function f constructs a complete graph.

is-complete (f G)

Given Hamiltonian cycle T, the function g selects, for each edge e of G, an endpoint to be included in the result. Thus, the function g computes a vertex cover for G given a Hamiltonian cycle for H.

$$\textit{is-hc}\ (f\ G)\ T \Longrightarrow \textit{is-vc}\ G\ (g\ G\ T)$$

Next, I describe the proofs for the linear inequalities that are required for the L-reduction. But first, I prove two necessary lemmas.

We prove an upper bound for the total cost of the optimal Hamiltonian cycle for the graph H. This lemma corresponds to the property (a) that is used by the L-reduction proof in [9, Theorem 21.1]. Let OPT_H be an optimal Hamiltonian cycle for H and let OPT_{VC} be an optimal vertex cover for G. An upper bound for the total cost of OPT_H is given by the following inequality.

$$\textit{cost-of-path}_c\ OPT_H \leq 15 * |E(G)| + |OPT_{VC}|$$

Intuitively, we construct a Hamiltonian cycle T for H from the optimal vertex cover OPT_{VC} for G. The total cost of T is at least the total cost of the optimal Hamiltonian cycle OPT_H. The Hamiltonian cycle T traverses each subgraph H_e with the Hamiltonian path that starts and ends at the corner vertices that correspond to the covering vertex of e in the vertex cover OPT_{VC}. There are $|E(G)|$ many subgraph H_e and each Hamiltonian path for a subgraph H_e has a total cost of 11. We now need to add $|E(G)|$ many edges to connect the start- and end-vertices of the Hamiltonian paths to form a Hamiltonian cycle for H. We pick as many edges with weight 4 ("cheaper" edges) as possible. For the remaining connections, we use "expensive" edges with a weight of 5. It turns out we can select at most $|E(G)| - |OPT_{VC}|$ many edges of weight 4. Thus, the total weight of the constructed Hamiltonian cycle T is bounded by the following expression.

$$\begin{aligned} 11 * |E(G)| + 4 * (|E(G)| - |OPT_{VC}|) + 5 * |OPT_{VC}| \\ = 15 * |E(G)| + |OPT_{VC}| \end{aligned}$$

Essentially, the number of "expensive" edges in the Hamiltonian cycle T for H corresponds to the cardinality of the vertex cover for G. The generalization of this construction for an arbitrary vertex cover X is formalized by the following lemma.

$$\textit{is-vc}\ G\ X \Longrightarrow \exists\, T.\ \textit{is-hc}\ (f\ G)\ T \wedge \textit{cost-of-path}_c\ T \leq 15 * |E(G)| + |X|$$

Another property that is required for the proof of the linear inequalities of the L-reduction is an upper bound for the cardinality of the vertex cover that is constructed by the function g. The following lemma corresponds to the property (c) that is used by the L-reduction proof in [9, Theorem 21.1].

$$\textit{is-hc}\ (f\ G)\ T \Longrightarrow \exists\, k{\geq}|g\ G\ T|.\ 15 * |E(G)| + k \leq \textit{cost-of-path}_c\ T$$

The cardinality of the vertex cover that is computed by g can be at most the number of "expensive" edges in a Hamiltonian cycle T of H. This follows from the fact that any two edges e and f in G ($e \neq f$), where the subgraphs H_e and H_f are connected by a "cheap" edge in T, are covered by the same endpoint in the vertex cover $g\ G\ T$. Thus, there can only be different covering vertices if two subgraphs are connected by an "expensive" edge.

With the properties proved above, we prove the first linear inequality required for the L-reduction. By assumption, the degree of every vertex in G is at most 4. Thus, the number of edges of G is at most 4-times the cardinality of any vertex cover of G, in particular the optimal vertex cover OPT_{VC} of G. The following chain of inequalities proves the first inequality required for the L-reduction.

$$\begin{aligned}\textit{cost-of-path}_c\ OPT_H &\leq 15 * |E(G)| + |OPT_{VC}| \\ &\leq 15 * (4 * |OPT_{VC}|) + |OPT_{VC}| \\ &\leq 61 * |OPT_{VC}|\end{aligned}$$

The second inequality for the L-reduction follows from the upper-bound for the cardinality of the vertex cover that is computed by g and the upper-bound for the total cost of the optimal Hamiltonian cycle OPT_H. The following chain of inequalities proves the second linear inequality for the L-reduction.

$$\begin{aligned}||g\ G\ T| - |OPT_{VC}|| &\leq |k - |OPT_{VC}|| \\ &= |15 * |E(G)| + k - 15 * |E(G)| - |OPT_{VC}|| \\ &\leq |\textit{cost-of-path}_c\ T - \textit{cost-of-path}_c\ OPT_H|\end{aligned}$$

5.3 Incompleteness and Unfinished Business

In this section, I point out two cases that are not covered in the L-reduction proof by [9, Theorem 21.1] that I discovered during the formalization. I also describe the parts of my formalization that are not finished.

The L-reduction proof that is presented in [9, Theorem 21.1] is incomplete and does not cover the following cases.

(i) The proof by [9, Theorem 21.1] fails if the optimal vertex cover of G has a cardinality of 1. In this case, the upper-bound for optimal Hamiltonian cycle for H does not hold. The construction for the L-reduction still works in this case, but the proofs for this case need to be considered separately. I circumvent this problem by explicitly excluding this case through an additional assumption. Formalizing the proof for this case should be straightforward.

(ii) The proof for [9, Theorem 21.1] fails to cover all cases when constructing the vertex cover for G from a Hamiltonian cycle T for H (function g). There are different types of Hamiltonian paths for the subgraphs H_e that do not start or end at corner vertices of the subgraph H_e (see Figures 2). In [9], it is only described how to pick a covering vertex if the Hamiltonian path for the corresponding subgraph H_e starts and ends at a corner vertex of the subgraph H_e. For some Hamiltonian paths that do not start or end at corners of the subgraph H_e, it is not immediately clear which covering vertex to choose. Nevertheless, I have solved this issue by extending the definition of the function g. The definition of the function g described in [9, Theorem 21.1] only replaces parts of the given tour T with a Hamiltonian path for a subgraph H_e when the tour T does not already contain a Hamiltonian path for the subgraph H_e. Thus, the resulting tour may contain Hamiltonian paths for the subgraphs H_e that do not start or end at corner vertices of the subgraph H_e. Therefore, the construction of the vertex cover for G from the modified tour T' would require an extra case distinction to handle the different types of Hamiltonian paths for the subgraphs H_e. The required case distinction is not made in [9]. The types of Hamiltonian paths for the subgraph H_e depicted in the Figures 2b and 2c are not covered. On the other hand, I defined the function g using the function *shorten-tour* which explicitly handles these cases.

The proof in [9, Theorem 21.1] references a proof by [13]. The proof by [13] uses a similar construction to prove the MaxSNP-harness of a restricted version of the Metric TSP, where the values of the edge weights are restricted to only 1 or 2. Because of the restricted edge weights, the proof by [13] does not have this issue.

Two aspects make my formalization of the L-reduction incomplete. Firstly, I have not finished the proof that shows that the subgraph H_e only admits the three types of Hamiltonian paths depicted in Figure 2. The theory `./reductions/FindHamiltonianPath.thy` contains an instantiation of a suitable lemma with an executable graph representation. By exhaustive search, the instantiated lemma is proven. This result needs to be transferred to the reduction proof. At the moment I am not sure what the best way to do this is. Previously, I have also attempted to prove this by coming up with an abstract lemma about paths in the subgraph H_e. Using this abstract lemma, my goal was to prove that there is no Hamiltonian path that starts and ends at vertices on opposing sides of the subgraph. Unfortunately, I did not have success with this approach.

Secondly, I have not formally proven the polynomial running times of the functions f and g. To formally reason about running times a formal computational model is required. The imperative language `IMP-` provides such

a computational model. A possible approach is to refine instantiations of the functions f and g to `IMP-`. Ultimately, the running times of the refined functions would be proven.

6 Future Work and Conclusion

In this work, I present my formalization of parts of section 21.1, *Approximation Algorithms for the TSP* from [9] with the interactive theorem prover Isabelle/HOL. I continue the work of [6] by formalizing and formally verifying two approximation algorithms for Metric TSP: the DoubleTree and the Christofides-Serdyukov algorithm. For that, I build on the formalization of graphs by [1]. I formalize many graph-related concepts in Isabelle/HOL, such as weighted graphs, Eulerian Tours, or Hamiltonian cycles. Moreover, I formalize an L-reduction from the VCP4 to Metric TSP. Thereby, I present an approach to formalize an L-reduction in Isabelle/HOL. To my knowledge, this is the first formalization of an L-reduction in Isabelle/HOL. A consequence of the L-reduction is that the Metric TSP is MaxSNP-hard and thus there cannot be an approximation scheme for the Metric TSP unless P = NP.

To formalize the L-reduction it is necessary to verify executable functions on graphs. For that, I use an abstract graph representation that is based on an adjacency map. The graph representation is instantiated with a concrete implementation to obtain executable functions on graphs. This approach is versatile and can be applied when reasoning about executable functions on graphs.

Future work includes proving the existence of the necessary algorithms for the DoubleTree and Christofides-Serdyukov algorithm. Therefore, the formalizations of algorithms to compute a MST, a minimum perfect matching, and an Eulerian tour are needed. There is already an existing formalization of Prim's algorithm in Isabelle/HOL [10]. Given a graph, Prim's algorithm computes a MST. In the theory `./adaptors/BergePrimAdaptor.thy`, I have started (but not finished) to implement an adaptor between my formalization and the formalization by [10]. Suitable algorithms for the other problems are e.g. Edmonds' Blossom algorithm [5] for minimum perfect matching and Euler's algorithm [9] for Eulerian tour.

Furthermore, the Eulerian graphs that are constructed for the DoubleTree and the Christofides-Serdyukov algorithm generally are multigraphs. Therefore, a formalization of multigraphs is required. In the theory `./graphs/MultiGraph.thy`, I have started (but not finished) to formalize an encoding of multigraphs with simple graphs.

For the L-reduction, it is required to prove the polynomial running time of the reduction functions f and g. This can be done by refining the functions

to a programming language that provides a computational model, e.g. IMP- [17]. The running times are then proven with the refined functions.

References

[1] M. Abdulaziz. Formalization of Berge's theorem. GitHub Repository, 2020. https://github.com/wimmers/archive-of-graph-formalizations/blob/master/Undirected_Graphs/Berge.thy.

[2] C. Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences of the United States of America*, 43(9):842–844, 1957.

[3] B. Brumitt and A. Stentz. Dynamic mission planning for multiple mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 3, pages 2396–2401 vol.3, 1996.

[4] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[5] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[6] R. Eßmann, T. Nipkow, and S. Robillard. Verified approximation algorithms. In N. Peltier and V. Sofronie-Stokkermans, editors, *Automated Reasoning*, pages 291–306, Cham, 2020. Springer International Publishing.

[7] M. Grötschel, M. Jünger, and G. Reinelt. Optimal control of plotting and drilling machines: A case study. *Zeitschrift für Operations Research*, 35(1):61–84, Jan 1991.

[8] A. R. Karlin, N. Klein, and S. O. Gharan. A (slightly) improved approximation algorithm for metric tsp. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 32–45, New York, NY, USA, 2021. Association for Computing Machinery.

[9] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Berlin, Heidelberg, 6th edition, 2018.

[10] P. Lammich and T. Nipkow. Purely functional, simple, and efficient implementation of Prim and Dijkstra. *Archive of Formal Proofs*, June 2019. https://isa-afp.org/entries/Prim_Dijkstra_Simple.html, Formal proof development.

[11] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[12] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991.

[13] C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18(1):1–11, Feb 1993.

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

[15] A. I. Serdyukov. On some extremal tours in graphs (translated from russian). *Upravlyaemye Sistemy (in Russian)*, 17:76–79, 1978.

[16] V. V. Vazirani. *Approximation Algorithms*. Springer Berlin, Heidelberg, 2013.

[17] S. Wimmer. Polynomial-time reductions in Isabelle/HOL. GitHub Repository, 2021. https://github.com/wimmers/poly-reductions.