Formalizing Theory of Approximating the Traveling-Salesman Problem

Lukas Koller

May 25, 2023

Abstract

The Traveling Salesman Problem (TSP) is a well-known NP-complete optimization problems. The Metric TSP is a simplification of the TSP where it is assumed that the input graph is complete and the edge weights satisfy the triangle-inequality. In this report, I present my formalization and formal verification of two approximation algorithm for the Metric TSP: the DoubleTree and Christofides-Serdyukov algorithm. I also describe the formalization of an 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. The TSP describes the problem of finding a tour with minimum total weight in a given weighted undirected graph s.t. 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]. 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 [17]. An approximation algorithm is a polynomial time algorithm that returns a bounded approximate solution. The approximation ratio gives a bound on the distance between the approximate solution and the optimal solution. Ideally, the approximation ratio is a constant factor. An approximation algorithm essentially trades the optimality of the returned solution for a polynomial running time. Unfortunately, there are bad news for the TSP: there is no constant-factor approximation algorithm for the TSP [15].

The Metric TSP is a simplifies the TSP by making 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 [15].

This report describes the formalization of parts of the section 21.1, *Approximation Algorithms for the TSP* from [9].

1. I have formalized and verified two approximation algorithms, DoubleTree and Christofides-Serdyukov, for Metric TSP.

2. I have formalized 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.

The MaxSNP-hardness of a problem proves the non-existence of an approximation scheme [12]. An approximation scheme is an approximation algorithm that takes an additional parameter $\varepsilon > 0$ and has 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]. A linear reduction (L-reduction) is special type of reduction for optimization problems [12], 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.

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

2 Related Work

With an approximation ratio of $\frac{3}{2}$, the Christofides-Serdyukov algorithm [4, 16] is the best known approximation algorithm for the Metric TSP [9]. Recently, [8] have developed a randomized approximation algorithm for the Metric TSP which (in expectation) achieves a slightly better approximation ratio than the Christofides-Serdyukov algorithm.

In [6] 6 different approximation algorithms are formalized and verified in Isabelle/HOL. I have followed their formalization approaches and formalized the DoubleTree and the Christofides-Serdyukov algorithm in similar fashion.

In [18] polynomial time reductions are formalized in Isabelle/HOL. To formally reason about running times of functions a computational model for the programming language is required. Therefore, [18] have developed the imperative language `IMP-` which provides a computational model to reason about the running times of function implemented in `IMP-`.

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 formalized as 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 E of a graph that satisfies the invariant *graph-invar*. The graph formalization by [1] provides definitions for the following concepts.

- Vertices of a graph: *Vs*
- Paths: *path*, *walk-betw*, and *edges-of-path*
- Degree of vertices: *degree*

- Connected components: *connected-component*
- Matchings: *matching*

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

All of the graph-related definitions are defined in theories which 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 is connected if all vertices are contained in the same connected component.

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

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 $\textit{cost-of-path}_c$ computes the cost of a given path with the edge weights c.

3.3 Complete Graphs

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

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

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

In a complete graph, any sequence of vertices s.t. no two consecutive vertices are equal is a path.

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

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 resulting path has a 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 is a simple if no edge is used twice.

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

A cycle is a path s.t. (i) the first and last vertex are the same, (ii) the path is simple, and (iii) the path 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 is acyclic if it does not contain a cycle.

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

A vertex that is contained in a cycle 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 is a graph that is (i) connected, (ii) acyclic.

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

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

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

A minimum spanning tree (MST) is a spanning tree with minimum total weight. The function $cost\text{-}of\text{-}st_c$ computes the total weight of a given spanning tree with the edge weights c.

$is\text{-}mst\ E\ c\ T \equiv is\text{-}st\ E\ T \wedge (\forall\, T'.\ is\text{-}st\ E\ T' \longrightarrow cost\text{-}of\text{-}st_c\ T \leq cost\text{-}of\text{-}st_c\ T')$

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

3.6 Hamiltonian Cycles

A Hamiltonian cycle is a non-empty tour in a graph that visits every vertex exactly once.

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

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

3.7 Traveling-Salesman Problem

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

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

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 regular graph.

3.9 Eulerian Tours

A graph 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* return the set of vertices of a multigraph.

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

A Eulerian tour is a path that uses every edge of a given graph exactly once. *mpath* is the predicate for a path in a multigraph.

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

The locale *eulerian* fixes an algorithm (denoted by *comp-et*) to compute a Eulerian tour for a given multi-graph.

3.10 Perfect Matchings

A perfect matching is a matching that contains every vertex.

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

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

$is\text{-}min\text{-}match\ E\ c\ M \equiv$
$is\text{-}perf\text{-}match\ E\ M \wedge$
$(\forall\, M'.\ is\text{-}perf\text{-}match\ E\ M' \longrightarrow cost\text{-}of\text{-}match_c\ M \leq cost\text{-}of\text{-}match_c\ M')$

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

3.11 Vertex Cover

A vertex cover is a subset of edges that contains at least one endpoint of every edge.

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

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

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

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

$[\![\bigwedge v.\ v \in Vs\ E \Longrightarrow degree\ E\ v \leq enat\ k;\ is\text{-}vc\ E\ X]\!] \Longrightarrow |E| \leq k * |X|$

4 Approximating the Metric TSP

This section describes the formalization of the DoubleTree and the Christofides-Serdyukov approximation algorithms for 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 total weight of at most the total weight of all edges in E'.

The proof for this proposition is constructive: first compute a 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"). The edge weights c satisfy the triangle-inequality (by assumption), 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 locale *hc-of-et* (see theory `./algorithms/DoubleTree.thy`) extends the locales *metric-graph-abs* and *eulerian* and thus assumes the necessary assumptions for the input graph 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 instance of an edge) of the graph E.

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

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 edges of the Eulerian tour P.

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

With the proposition [9, Lemma 21.3] the task of approximating Metric TSP boils down to constructing a 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 influcences the approximation-ratio.

Both approximation algorithms, DoubleTree and Christofides-Serdyukov, first compute a MST T of the input graph E. Then edges are added to the MST T to construct a Eulerian graph that spans the vertices of the graph E.

The DoubleTree algorithm constructs a Eulerian graph by simply doubling every edge of the MST T. In this multigraph every vertex has even degree and thus this multi-graph is a Eulerian graph. The total weight of any Hamiltonian cycle is at least the the total weight of T. Thus, the total weight of the Hamiltonian cycle produced by the DoubleTree algorithm is at most 2-times 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 M and the MST T is a Eulerian graph. 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 main steps.

1. Compute a MST T of the input graph E.
2. Compute a 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 to 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 locale *double-tree-algo* extends the locales *hc-of-et* and *mst* and thus assumes the existence of the required algorithms. The existence of a function *comp-et* to compute an Eulerian tour is already is assumed by the locale *hc-of-et*. The function *comp-mst* denotes the assumed function to compute a MST.

```
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 need to be 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*.

$$\textit{cost-of-path}_c\ \textit{double-tree} \leq (2 :: 'b) * \textit{cost-of-path}_c\ \textit{OPT}$$

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

$$\begin{array}{l}
\textit{VARS}\ T\ T_{2x}\ v\ P\ P'\ H \\
\ \{\ \textit{True}\ \} \\
\quad T := \textit{comp-mst}\ c\ E; \\
\quad T_{2x} := \textit{mset-set}\ T + \textit{mset-set}\ T; \\
\quad P := \textit{comp-et}\ T_{2x}; \\
\quad P' := P; \\
\quad H := []; \\
\quad \textit{WHILE}\ P' \neq [] \\
\quad \textit{INV}\ \{\ \textit{comp-hc-of-et}\ P\ [] = \textit{comp-hc-of-et}\ P'\ H \\
\qquad \wedge\ P = \textit{comp-et}\ T_{2x} \\
\qquad \wedge\ T_{2x} = \textit{mset-set}\ T + \textit{mset-set}\ T \\
\qquad \wedge\ T = \textit{comp-mst}\ c\ E \\
\quad \}\ \textit{DO} \\
\qquad v := \textit{hd}\ P'; \\
\qquad P' := \textit{tl}\ P'; \\
\qquad \textit{IF}\ v \in \textit{set}\ H \wedge P' \neq []\ \textit{THEN} \\
\qquad\quad H := H \\
\qquad \textit{ELSE} \\
\qquad\quad H := v\#H \\
\qquad \textit{FI} \\
\quad \textit{OD} \\
\ \{\ \textit{is-hc}\ E\ H \wedge \textit{cost-of-path}_c\ H \leq 2 * \textit{cost-of-path}_c\ \textit{OPT}\ \}
\end{array}$$

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 3 algorithms: the 2 algorithms the DoubleTree algorithm dependes 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 in a similar fashion to the DoubleTree algorithm. The locale *christofides-serdyukov-algo* extends the locale *double-tree-algo* and adds the assumption for the existence of an algorithm to compute a minimum perfect-matching. The function *comp-match* computes a minimum perfect-matching for a given graph. The definition of the Christofides-Serdyukov algorithm in the locale *christofides-serdyukov-algo* is straightforward.

```
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 :: 'b) * cost-of-path_c christofides-serdyukov
≤ (3 :: 'b) * 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.

```
VARS T W M J v P P' H
 { 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 $P' \neq []$
INV { *comp-hc-of-et* P $[]$ = *comp-hc-of-et* P' H
$\wedge$ P = *comp-et* J
$\wedge$ J = *mset-set* T + *mset-set* M
$\wedge$ M = *comp-match* $(\{e \in E.\ e \subseteq W\})$ c
$\wedge$ $W = \{v \in Vs\ T.\ \neg\ even'\ (degree\ T\ v)\}$
$\wedge$ T = *comp-mst* c E
} *DO*
$v := hd\ P'$;
$P' := tl\ P'$;
IF $v \in set\ H \wedge P' \neq []$ *THEN*
$H := H$
ELSE
$H := v\#H$
FI
OD
{ *is-hc* E $H \wedge 2 * cost\text{-}of\text{-}path_c\ H \leq 3 * cost\text{-}of\text{-}path_c\ OPT$ }

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 of 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. A vertex-cover is subset of vertices that contains at least one end-point of every edge. 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 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 and $OPT\ (f\ x_A) \leq \alpha(OPT\ x_A)$, and

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

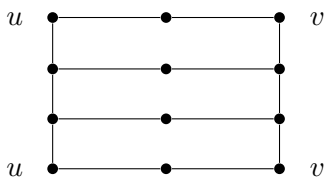
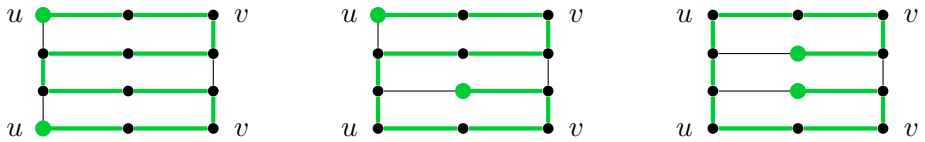


Figure 1: Subgraph H_e for an edge $e{:=}\{u,v\}$. Each corner vertex of H_e corresponds to an endpoint of e.



(a) (b) (c)

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

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.

We L-reduce the VCP4 to the Metric TSP by defining the functions f and g and proving their required properties. 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 construct 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). The subgraph H_e consists of 12 vertices that are arranged in a 4-by-3 lattice structure. Each corner vertex of subgraph H_e corresponds to an endpoint of the edge e. The 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 in 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 a subgraph H_e. We distinguish between 3 types of edges.

(i) An edge that connects two vertices which both belong to the same subgraph H_e has the 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 vertices correspond to the same endpoint has the weight of 4.

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

The proof that the edge weights satisfy the traingle-inequality requires a long and tedious case distinction and thus is omitted.

Next, I describe the definition of the function g. The function g maps a Hamiltonian cycle T in 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 functions f and g are required to be computable in polynomial time. The locale *ugraph-adj-map* provides an abstract adjacency map which serves as a graph representation that allows for the implementation of executable function 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 necessary graph-related definitions are redefined for the adjencecy-map representation of graphs (see theory `./graphs/GraphAdjMap_Specs.thy`). The definitions for the adjencecy-map representation of graphs are denoted by the postfix $-Adj$.

The set of undirected edges of a graph which is represented by an adjacency map is returned by the function *uedges*. A linear order on the vertices is used to identify different instances of the same undirected edge.

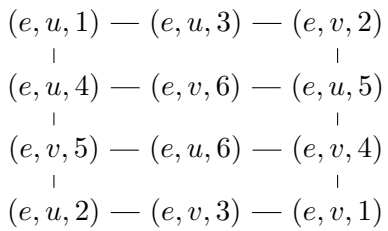


Figure 3: Formalization of the subgraph H_e for an edge e:=$\{u,v\}$.

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 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,u,i) where u is an endpoint of the edge e and $i \in \{1..6\}$ is an index. The 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 ($H_{\{u,v\}}$=$H_{\{v,u\}}$).

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.

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

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 of edges. To simplify things, the case for two vertices that are neighbours 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 required to compare the undirected edges.

$c\ G\ (e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2) =$
$(\mathsf{if}\ \mathit{is\text{-}edge\text{-}in\text{-}He}\ G\ (\mathit{uEdge}\ (e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2))\ \mathsf{then}\ 1$
$\ \mathsf{else\ if}\ \mathit{are\text{-}vertices\text{-}in\text{-}He}\ G\ (e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2)$
$\quad \mathsf{then}\ \mathit{the\text{-}enat}\ (\mathit{min\text{-}dist\text{-}in\text{-}He}\ G\ (e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2))$

else if $rep1\ e_1 \neq rep1\ e_2 \wedge$
$w_1 = w_2 \wedge (i_1 = 1 \vee i_1 = 2) \wedge (i_2 = 1 \vee i_2 = 2)$
then 4 *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 cycle 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.

$g\ G\ T = vc\text{-}of\text{-}tour\ G\ (tl\ (shorten\text{-}tour\ G\ T))$

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 appends a Hamiltonian path for the subgraph H_e. This replacement 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 appended Hamiltonian path for the subgraph H_e is carefully chosen such that the resulting tour has less total weight. The function *shorten-tour* formalizes property (b) from the L-reduction proof in [9, Theorem 21.1].

Contrary, the function g described by [9, Theorem 21.1] only replaces parts of the 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' requires an extra case distinction to handle the different types of Hamiltonian paths for the subgraphs H_e (see Figure 2). [9, Theorem 21.1] does not make the required case distinction in the definition of the function g. The types 2b and 2c of Hamiltonian paths for the subgraph H_e from Figure 2 are not covered. On the other hand, the function *shorten-tour* explicitly handles these cases.

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 a L-reduction.

The function f constructs a complete graph.

$g2.is\text{-}complete\text{-}Adj\ (f\ G)$

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.

$g2.is\text{-}hc\text{-}Adj\ (f\ G)\ T \Longrightarrow g1.is\text{-}vc\text{-}Adj\ G\ (g\ G\ T)$

Next, I describe the proofs for the linear inequalities that are required for the L-reduction. The following two lemmas formalize two important properties that are needed for the proofs of the linear inequalities.

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_{mTSP} be an optimal Hamiltonian cycle for H and let OPT_{VCP4} be an optimal vertex cover for G.

$cost\text{-}of\text{-}path\ (c\ G)\ OPT_{mTSP} \leq 15 * |g1.uedges\ G| + card1\ OPT_{VCP4}$

Intuitively, using the optimal vertex cover OPT_{VCP4} for G, a Hamiltonian cycle T for H is constructed. The total cost of T is at least the total cost of the optimal Hamiltonian cycle OPT_{mTSP}. 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_{VCP4}. Each Hamiltonian path for a subgraph H_e has a total cost of 11. We now need to add $|g1.uedges\ G|$ 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 weight 5. It turns out we can select at most $|g1.uedges\ G| - card1\ OPT_{VCP4}$ many edges of weight 4. Thus, the total weight of the constructed Hamiltonian cycle T is at most $11 * |g1.uedges\ G| + 4 * (|g1.uedges\ G| - card1\ OPT_{VCP4}) + 5 * card1\ OPT_{VCP4} = 15 * |g1.uedges\ G| + card1\ OPT_{VCP4}$. 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 is formalized by the following lemma.

$g1.is\text{-}vc\text{-}Adj\ G\ X \Longrightarrow$
$\exists\, T.\ g2.is\text{-}hc\text{-}Adj\ (f\ G)\ T \wedge$
$\quad cost\text{-}of\text{-}path\ (c\ G)\ T \leq 15 * |g1.uedges\ G| + card1\ X$

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. This lemma corresponds to the property (c) that is used by the L-reduction proof in [9, Theorem 21.1].

$g2.is\text{-}hc\text{-}Adj\ (f\ G)\ T \Longrightarrow$
$\exists\, k{\geq}card1\ (g\ G\ T).\ 15 * |g1.uedges\ G| + k \leq cost\text{-}of\text{-}path\ (c\ G)\ T$

The cardinality of the vertex cover that is constructed by the function g can be at most the number "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. Thus, there can only be distinct covering vertices if two subgraphs are connected by an "expensive" edge.

With the lemmas proved above, we prove the linear inequalities required for the L-reduction. The first linear inequality is formalized by the following lemma.

$$\textit{cost-of-path}\ (c\ G)\ OPT_{mTSP} \leq 61 * \textit{card1}\ OPT_{VCP4}$$

By assumption, the degree of every vertex in G is at most 4. Thus, the number of edges of G is bounded by 4-times the cardinality of any vertex cover of G. The first inequality is a consequence of the upper-bound on the total cost of the optimal Hamiltonian cycle OPT_{mTSP} for H and the upper-bound of the number of edges of G.
The following lemma formalizes the second linear inequality for the L-reduction.

$$\begin{aligned}&\textit{g2.is-hc-Adj}\ (f\ G)\ T \Longrightarrow\\ &|\textit{card1}\ (g\ G\ T) - \textit{card1}\ OPT_{VCP4}|\\ &\leq 1 * |\textit{cost-of-path}\ (c\ G)\ T - \textit{cost-of-path}\ (c\ G)\ OPT_{mTSP}|\end{aligned}$$

The second inequality follows from the upper-bound for the cardinality of the vertex cover that is constructed by the function g and the upper-bound for the total cost of the optimal Hamiltonian cycle OPT_{mTSP}.

5.3 Flaws and Unfinished Business

In this section, I describe two flaws in the L-reduction proof by [9, Theorem 21.1] that I discovered during the formalization. I also go over the parts of my formalization that are not finished.
The L-reduction proof that is presented in [9, Theorem 21.1] is incomplete. During the formalization of the L-reduction I have discovered two flaws in the proof by [9, Theorem 21.1].

(i) The proof by [9, Theorem 21.1] fails if the optimal vertex cover of G has a cardinality of 1. In this case, some of the auxiliary lemmas used in [9] do not hold. In particular 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 and just a matter of adapting the necessary lemmas.

(ii) The proof for [9, Theorem 21.1] does not cover every case, when identifying the covering vertices for G from a Hamiltonian cycle T for H. There are different types Hamiltonian paths for the subgraphs H_e that do not start or end at corner vertices of the subgraph H_e. [9] only describe how to identify a covering vertex if the Hamiltonian path 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. 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 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 (see Figure 2). [9] does not make the required case distinction in the definition of the function g. The types of Hamiltonian paths for the subgraph H_e depicted in the Figures 2b and 2c are not covered.

[9] cite a proof by [13]. The proof by [13] use 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.

On the other hand, I defined the function g using the function *shorten-tour* which explicitly handles these cases.

There are two aspects that make my formalization of the L-reduction incomplete. Firstly, I have not finished the proof that the subgraph H_e only admits three types of Hamiltonian paths (see Figure 2). The lemma is stated in the theory `reductions/VertexCover4ToMetricTravelingSalesman_Specs.thy`. The theory `./reductions/FindHamiltonianPath.thy` contains an instantiation this lemma for 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 and way to do this is. I have also attempted another way to prove this. I tried, by coming up with a clever naming of the vertices of the subgraph H_e to prove that there is no Hamiltonian path that start and ends at vertices on

opposing sides of the subgraph. Unfortunately, I did not have success with this approch.

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. `IMP-` was developed as part of a formalization of polynomial time reduction [18]. A possible approach to prove the polynomial running times of the functions f and g, is to refine instantiations of their definitions (with implementations for the locales *Map* and *Set2*) to `IMP-`. With the computational model provided by `IMP-`, we can prove the running times of the refined reduction functions.

6 Future Work and Conclusion

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

- I have formalized and formally verified two approximation algorithms for Metric TSP: the DoubleTree and the Christofides-Serdyukov algorithm.
- I have formalized an L-reduction from VCP4 to Metric TSP, which proves the MaxSNP-hardness of Metric TSP.

I have formalized a significant amount of graph-related concepts.

I present an approach to formalize of the L-reductions in Isabelle/HOL.

With the formalization and formal verification of the DoubleTree and the Christofides-Serdyukov algorithm I have continued the work of [6] in formalizing approximation algorithms in Isabelle/HOL.

Future work includes proving the existence of the necessary algorithms for the DoubleTree and Christofides-Serdyukov algorithm. Algorithms for the following problems is need to be formalized: MST, minimum perfect matching, and Eulerian tour. There is already an existing formalization of Prim's algorithm [10], an algorithm that computes a MST. In the theory `./adaptors/BergePrimAdaptor.thy`, I have started (but not finished) to implement an adaptor to 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 Christofides-Serdyukov algorithm might be multigraphs. Therefore, the 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 reduction functions f and g. Moreover, the proof that the subgraph H_e only admits certain Hamiltonian paths has to be finished.

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] S. Sahni and T. Gonzalez. P-complete approximation problems. *J. ACM*, 23(3):555–565, Jul 1976.

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

[17] V. V. Vazirani. *Approximation Algorithms*. Springer Publishing Company, Incorporated, 2010.

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