Formalizing Theory of Approximating the Traveling-Salesman Problem

Lukas Koller

June 10, 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 [4], logistics [18], and designing printed circuit boards [9].

The TSP is NP-hard [12, 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 [20] 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 trades 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 [12, 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 [12, Theorem 21.2].

In this work, I present my formalization of parts of section 21.1, *Approximation Algorithms for the TSP*, from [12].

1. I formalize and formally verify two approximation algorithms, DoubleTree and Christofides-Serdyukov, for the 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 [12, Corollary 16.40]. An approximation scheme takes a parameter $\varepsilon > 0$ and produces an approximation algorithm 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 [15, Theorem 1]. A linear reduction (L-reduction) [15] 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 [12, Theorem 16.46].

Moreover, to get started with the project and to familiarize myself with the existing formalization of undirected graphs by Abdulaziz [1], I have formalized a proof by Korte and Vygen [12, Proposition 11.1] which shows the equivalence of the maximum weight matching problem (Maximum Matching Problem) and the minimum weight perfect matching problem (Minimum Matching Problem).

All of my formalizations are done with the interactive theorem prover Isabelle/HOL [14]. My formalizations can be found on GitHub: https://github.com/kollerlukas/tsp.

2 Related Work

The Christofides-Serdyukov algorithm [5, 19] has an approximation ratio of $\frac{3}{2}$, and thus is the best known approximation algorithm for the Metric TSP [12]. 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 [11].

A couple of different approximation algorithms are already formalized and verified in Isabelle/HOL [8]. I continue the work of Eßmann et al. [8] by formalizing and verifying the DoubleTree and the Christofides-Serdyukov approximation algorithms. Similar to Eßmann et al., I give implementations for both of the formalized approximation algorithms with an imperative `WHILE`-program.

Wimmer [21] formalize polynomial time reductions in Isabelle/HOL. To formally reason about the running times of functions, a computational model for the programming language is required. Therefore, Wimmer has 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 Abdulaziz [1], which is developed for a formalization of Berge's theorem [3]. An undirected graph is represented by a finite set of edges. An edge is represented by a doubleton set. The following invariant characterizes a well-formed graph.

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

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* returns the connected component of a vertex in a graph.
- A matching is a graph where no two edges are incident to the same vertex. 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 the 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 any two 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 E 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 E, any sequence of vertices P s.t. no two consecutive vertices are equal (*distinct-adj*) is a path.

Lemma 3.1. $\textit{is-complete}\ E \wedge \textit{distinct-adj}\ P \wedge P \subseteq \textit{Vs}\ E \Longrightarrow \textit{path}\ E\ P$

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 along 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 in X. The following lemma proves that the resulting path has less total weight.

Lemma 3.2. $P \subseteq \textit{Vs}\ E \Longrightarrow \textit{cost-of-path}_c\ (\textit{short-cut}\ X\ P) \leq \textit{cost-of-path}_c\ P$

3.4 Acyclic Graphs

A path P is simple if the path does not traverse an edge twice. This definition is in accordance to [13].

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

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.

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

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

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

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

Lemma 3.3. $\textit{is-cycle } E\ C \wedge v \in C \Longrightarrow \mathit{2} \leq \textit{degree } E\ v$

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*) that computes a MST for a given connected graph. This locale can be instantiated with a formalization of a suitable algorithm e.g. the formalization of Prim's algorithm by Lammich and Nipkow [13]. Given a connected graph, Prim's algorithm [10, 17] 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 Lammich and Nipkow. The adaptor already connects both formalizations for undirected graphs and proves the equivalence of the required definitions, e.g. definitions for MST. The only missing piece is to prove that the instantiated function for Prim's algorithm indeed computes a MST. This should be straightforward given the equivalences of the required definitions are already proven.

3.6 Hamiltonian Cycles

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

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

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 simple path.

3.7 Traveling-Salesman Problem

The optimal solution H to the TSP is a Hamiltonian cycle for the graph E with minimum total weight.

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

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. explain more...

3.9 Eulerian Tours

A graph E is Eulerian if every vertex has even degree. The DoubleTree and the Christofides-Serdyukov algorithm need the concept of 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. This locale could be instantiated with a formalization of Eulers's algorithm [12].

3.10 Weighted Matchings and Perfect Matchings

A subgraph M is a maximum weight matching for a graph E with edge weights c if M is a matching and the total weight of M is maximum.

$$\begin{aligned}&\textit{is-max-match}\ E\ c\ M \equiv M \subseteq E \wedge \textit{matching}\ M\\ &\ \wedge (\forall M'.\ M' \subseteq E \wedge \textit{matching}\ M' \longrightarrow (\textstyle\sum e{\in}M'.\ c\ e) \le (\textstyle\sum e{\in}M.\ c\ e))\end{aligned}$$

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.

$\textit{is-min-match}\ E\ c\ M \equiv \textit{is-perf-match}\ E\ M$
$\wedge (\forall M'.\ \textit{is-perf-match}\ E\ M' \longrightarrow (\sum e{\in}M.\ c\ e) \leq (\sum e{\in}M'.\ c\ e))$

The locale *min-weight-matching* fixes an algorithm (denoted by *comp-match*) that computes a minimum perfect matching for a given graph. For an instantiation of this locale a formalization of e.g. Edmond's Blossom algorithm [7, 6] for minimum-weight perfect matching is required. TODO: check citation of algorithm

3.11 Vertex Cover

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

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

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 at most k-times the cardinality of any vertex cover X for E.

Lemma 3.4. $(\forall v{\in}\textit{Vs}\ E.\ \textit{degree}\ E\ v \leq \textit{enat}\ k) \wedge \textit{is-vc}\ E\ X \Longrightarrow |E| \leq k * |X|$

4 Equivalence of the Maximum Weight Matching Problem and the Minimum Weight Perfect Matching Problem

In this section, I go over my formalization of [12, Proposition 11.1] which shows the equivalence of the Maximum Matching Problem and the Minimum Matching Problem.

Similar to the formalization of polynomial time reductions by Wimmer [21], a problem P consists of a function *is-sol* that characterizes the solutions to instances of the problem. The function *is-sol* takes a problem instance and an a solution as arguments.

datatype $('a,\ 'b)\ \mathit{prob} = \mathit{Prob}\ ('a \Rightarrow\ 'b \Rightarrow \mathit{bool})$

I define the two matching problems: the Maximum Matching Problem and the Minimum Matching Problem. For a well-formed graph E with edge weights c, a solution to the Maximum Matching Problem is a maximum weight matching (*is-max-match*).

$P_{max} \equiv \mathit{Prob}\ (\lambda(E,\ c)\ M.\ \mathit{graph\text{-}invar}\ E \longrightarrow \mathit{is\text{-}max\text{-}match}\ E\ c\ M)$

If a given well-formed graph E with edge weights c admits a perfect matching, then a solution to the Minimum Matching Problem is a minimum weight perfect matching (*is-min-match*). Otherwise, the solution is *None*.

```
$P_{min} \equiv \mathit{Prob}\ (\lambda(E,c)\ M.\ \mathit{graph\text{-}invar}\ E \longrightarrow (\mathbf{case}\ M\ \mathit{of}$
    $\mathit{Some}\ M \Rightarrow \mathit{is\text{-}min\text{-}match}\ E\ c\ M$
  $|\ \mathit{None} \Rightarrow \nexists M.\ \mathit{is\text{-}perf\text{-}match}\ E\ M))$
```

If a function f returns a solution to every instance of a problem P, then the function f solves the problem P. The predicate *solves* characterizes functions that solve a particular problem.

$\mathit{solves}\ P\ f \equiv \forall\, a.\ \mathit{is\text{-}sol}\ P\ a\ (f\ a)$

Given a function g that solves the Maximum Matching Problem we want to solve the Minimum Matching Problem. Let E be an undirected graph with edge weights c. We change the edge weight of each edge as follows.

$$c'\ e = (1 :: 'a) + (\sum e{\in}E.\ |c\ e|) - c\ e$$

A maximum weight matching for the graph E with edge weights c' is a minimum weight perfect matching for the graph E with edge weights c. The function f_1 solves the problem P_{min} and takes as an argument a function g that solves the problem P_{max}.

```
$f_1\ g\ (E,c) \equiv \mathit{let}\ M = g\ (E,\lambda e.\ 1 + (\sum e{\in}E.\ |c\ e|) - c\ e)\ \mathit{in}$
  $\mathit{if}\ \mathit{Vs}\ M = \mathit{Vs}\ E\ \mathit{then}\ \mathit{Some}\ M\ \mathit{else}\ \mathit{None}$
```

Given a function g that solves the Minimum Matching Problem we want to solve the Maximum Matching Problem. Let E be an undirected graph with edge weights c. We construct a graph H by doubling the graph E and adding an edge to connect each two copies of the same vertex. The function $E_H\ E$ construct this graph.

```
$E_H\ E \equiv$
  $\{\{(v,i),(w,i)\}\ |v\ w\ i.\ \{v,w\} \in E \wedge i{\in}\{1,2\}\} \cup \{\{(v,1),(v,2)\}\ |v.\ v \in \mathit{Vs}\ E\}$
```

The function c_H defines the edge weights for the constructed graph H. The cost of an edge in the first copy is its negative original weight. All other edges have no cost

$c_H\ (E,c)\ e \equiv \mathit{if}\ \mathit{fst}\ `\ e \in E \wedge \mathit{snd}\ `\ e = \{1\}\ \mathit{then}\ -c\ (\mathit{fst}\ `\ e)\ \mathit{else}\ 0$

With the function g, we compute a minimum weight perfect matching M for the graph H with edge weights c_H. The matching M is a maximum weight matching for the graph E with edge weights c. The function f_2 solves the problem P_{max} and takes as an argument a function g that solves the problem P_{min}.

$f_2\ g\ (E,c) \equiv \mathit{case}\ g\ (E_H\ E, c_H\ (E,c))\ \mathit{of}$
  $\mathit{Some}\ M \Rightarrow \{\{v,w\}\ |v\ w.\ \{(v,(1::\mathit{nat})),(w,1)\} \in M\}$

The main results of the formalization of [12, Proposition 11.1] are the following two theorems.

Theorem 4.1. $\mathit{solves}\ P_{max}\ g \Longrightarrow \mathit{solves}\ P_{min}\ (f_1\ g)$

Theorem 4.2. $\mathit{solves}\ P_{min}\ g \Longrightarrow \mathit{solves}\ P_{max}\ (f_2\ g)$

5 Approximaton Algorithms for 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 lemma.

Lemma 5.1 ([12, 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 has the same vertices as 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'.*

Proof. The proof 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, and 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'. □

illustration?

The construction for Lemma 5.1 is formalized with the function *comp-hc-of-et*. Given an Eulerian tour P 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.

$\mathit{comp\text{-}hc\text{-}of\text{-}et}\ []\ H = H$
$\mathit{comp\text{-}hc\text{-}of\text{-}et}\ [u]\ H = u \cdot H$
$\mathit{comp\text{-}hc\text{-}of\text{-}et}\ (u \cdot v \cdot P)\ H =$
$(\textsf{if}\ u \in H\ \textsf{then}\ \mathit{comp\text{-}hc\text{-}of\text{-}et}\ (v \cdot P)\ H\ \textsf{else}\ \mathit{comp\text{-}hc\text{-}of\text{-}et}\ (v \cdot P)\ (u \cdot H))$

The function *comp-hc-of-et* can be restated using the function *remdups* which removes duplicates from a list.

$\mathit{comp\text{-}hc\text{-}of\text{-}et}\ P\ [] =$
$(\textsf{if}\ 0 < |P|\ \textsf{then}\ \mathit{last}\ P \cdot \mathit{remdups}\ (\mathit{rev}\ (\mathit{butlast}\ P))\ \textsf{else}\ [])$

It is assumed that the multigraph E' has the same vertices as 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.

Lemma 5.2. $\mathit{is\text{-}et}\ E'\ P \wedge \mathit{mVs}\ E' = \mathit{Vs}\ E \wedge \mathit{set\text{-}mset}\ E' \subseteq E$
$\Longrightarrow \mathit{is\text{-}hc}\ E\ (\mathit{comp\text{-}hc\text{-}of\text{-}et}\ P\ [])$

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

Lemma 5.3. $\mathit{set}\ P \subseteq \mathit{Vs}\ E$
$\Longrightarrow \mathit{cost\text{-}of\text{-}path}_c\ (\mathit{comp\text{-}hc\text{-}of\text{-}et}\ P\ []) \leq \mathit{cost\text{-}of\text{-}path}_c\ P$

With Lemma 5.1, the task of approximating the 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 has the same vertices as the graph E. The function *comp-hc-of-et* is then applied to E' to obtain a Hamiltonian cycle for E.

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

5.1 Formalizing the DoubleTree Algorithm

The DoubleTree algorithm constructs an Eulerian graph by simply doubling every edge of a MST T for the input graph E. Thus, the DoubleTree algorithm consists of three steps.

1. Compute a MST T for the input graph E.
2. Compute an Eulerian tour P for 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 and
2. an algorithm to compute an Eulerian tour in an Eulerian multigraph.

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.

Theorem 5.4. *is-hc E double-tree*

The following theorem shows that the approximation ratio of the DoubleTree algorithm is 2. The optimal tour for the graph E is denoted with OPT.

Theorem 5.5. $\textit{cost-of-path}_c\ \textit{double-tree} \leq 2 * \textit{cost-of-path}_c\ OPT$

The total weight of any Hamiltonian cycle is at least the total weight of the MST 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.

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

```
$\{(\forall E\ c.\ \textit{is-connected}\ E \longrightarrow \textit{is-mst}\ E\ c\ (\textit{comp-mst}\ c\ E)) \wedge$
 $(\forall E.\ \textit{is-eulerian}\ E \longrightarrow \textit{is-et}\ E\ (\textit{comp-et}\ E))\}$
$T := \textit{comp-mst}\ c\ E;$
$T_2 := \textit{mset-set}\ T + \textit{mset-set}\ T;$
$P := \textit{comp-et}\ T_2;$
$P' := P;$
$H := [];$
$\textit{WHILE}\ P' \neq []$
$\textit{INV}\ \{\textit{comp-hc-of-et}\ P\ [] = \textit{comp-hc-of-et}\ P'\ H \wedge$
        $P = \textit{comp-et}\ T_2 \wedge$
        $T_2 = \textit{mset-set}\ T + \textit{mset-set}\ T \wedge$
        $T = \textit{comp-mst}\ c\ E \wedge$
        $(\forall E\ c.\ \textit{is-connected}\ E \longrightarrow \textit{is-mst}\ E\ c\ (\textit{comp-mst}\ c\ E)) \wedge$
        $(\forall E.\ \textit{is-eulerian}\ E \longrightarrow \textit{is-et}\ E\ (\textit{comp-et}\ E))\}$
$\textit{VAR}\ \{0\}$
$\textit{DO}\ v := \textit{hd}\ P';$
      $P' := \textit{tl}\ P';\ \textit{IF}\ v \in H \wedge P' \neq []\ \textit{THEN SKIP ELSE}\ H := v \cdot H\ \textit{FI}$
$\textit{OD}$
$\{\textit{is-hc}\ E\ H \wedge \textit{cost-of-path}_c\ H \leq 2 * \textit{cost-of-path}_c\ \textit{OPT}\}$
```

5.2 Formalizing the Christofides-Serdyukov Algorithm

The Christofides-Serdyukov algorithm is similar to the DoubleTree. Instead of doubling a MST T for the input graph E, 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. Therefore, the Christofides-Serdyukov algorithm consists of the following steps.

1. Compute a MST T for 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 for 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.

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.

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

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.

Theorem 5.6. *is-hc E christofides-serdyukov*

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

Theorem 5.7. $2 * cost\text{-}of\text{-}path_c\ christofides\text{-}serdyukov \leq 3 * cost\text{-}of\text{-}path_c\ OPT$

Let T be a MST for the input graph E and let W denote the vertices with odd degree in the MST T. The total weight of the Hamiltonian cycle that is computed by the Christofides-Serdyukov algorithm has at most the total weight of the MST T plus the total weight of the minimum weight perfect matching M between the vertices W. The total weight of the matching M is at most half the total weight of the optimal Hamiltonian cycle. The number of vertices in W is even, thus the edges of any Hamiltonian cycle for the vertices W can be split into two perfect matchings M_1 and M_2. Therefore, the total weight of the minimum weight perfect matching M is at most half the total weight of a optimal Hamiltonian cycle for the vertices W which is at most the total weight of the optimal Hamiltonian cycle for the input graph E. Therefore, the Christofides-Serdyukov algorithm is a $\frac{3}{2}$-approximation algorithm for the Metric TSP.

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

```
$\{(\forall E\ c.\ is\text{-}connected\ E \longrightarrow is\text{-}mst\ E\ c\ (comp\text{-}mst\ c\ E)) \wedge$
 $(\forall E.\ is\text{-}eulerian\ E \longrightarrow is\text{-}et\ E\ (comp\text{-}et\ E)) \wedge$
 $(\forall E\ c.\ (\exists M.\ is\text{-}perf\text{-}match\ E\ M) \longrightarrow is\text{-}min\text{-}match\ E\ c\ (comp\text{-}match\ E\ c))\}$
$T := comp\text{-}mst\ c\ E;$
$W := \{v \in Vs\ T \mid \neg\ even'\ (degree\ T\ v)\};$
$M := comp\text{-}match\ \{e \in E \mid e \subseteq W\}\ c;$
$J := mset\text{-}set\ T + mset\text{-}set\ M;$
$P := comp\text{-}et\ J;$
$P' := P;$
$H := [];$
```

```
$\mathit{WHILE}\ P' \neq []$
$\mathit{INV}\ \{\mathit{comp\text{-}hc\text{-}of\text{-}et}\ P\ [] = \mathit{comp\text{-}hc\text{-}of\text{-}et}\ P'\ H\ \wedge$
          $P = \mathit{comp\text{-}et}\ J\ \wedge$
          $J = \mathit{mset\text{-}set}\ T + \mathit{mset\text{-}set}\ M\ \wedge$
          $M = \mathit{comp\text{-}match}\ \{e \in E \mid e \subseteq W\}\ c\ \wedge$
          $W = \{v \in \mathit{Vs}\ T \mid \neg\ \mathit{even}'\ (\mathit{degree}\ T\ v)\}\ \wedge$
          $T = \mathit{comp\text{-}mst}\ c\ E\ \wedge$
          $(\forall E\ c.\ \mathit{is\text{-}connected}\ E \longrightarrow \mathit{is\text{-}mst}\ E\ c\ (\mathit{comp\text{-}mst}\ c\ E))\ \wedge$
          $(\forall E.\ \mathit{is\text{-}eulerian}\ E \longrightarrow \mathit{is\text{-}et}\ E\ (\mathit{comp\text{-}et}\ E))\ \wedge$
          $(\forall E\ c.\ (\exists M.\ \mathit{is\text{-}perf\text{-}match}\ E\ M) \longrightarrow \mathit{is\text{-}min\text{-}match}\ E\ c\ (\mathit{comp\text{-}match}\ E\ c))\}$

$\mathit{VAR}\ \{0\}$
$\mathit{DO}\ v := \mathit{hd}\ P';$
      $P' := \mathit{tl}\ P';\ \mathit{IF}\ v \in H \wedge P' \neq []\ \mathit{THEN\ SKIP}\ \ \mathit{ELSE}\ H := v \cdot H\ \mathit{FI}$
$\mathit{OD}$
$\{\mathit{is\text{-}hc}\ E\ H \wedge 2 * \mathit{cost\text{-}of\text{-}path}_c\ H \leq 3 * \mathit{cost\text{-}of\text{-}path}_c\ \mathit{OPT}\}$
```

6 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 [12, Theorem 21.6].

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 [12, 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 $\alpha,\beta > 0$ 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.

  $|(c_A\ x_A\ (g\ x_A\ y_B)) - OPT\ x_A| \leq |(c_A\ (f\ x_A)\ y_B) - OPT\ (f\ x_A)|$.

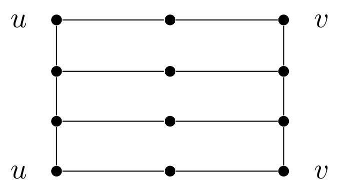

Figure 1: Subgraph H_e for an edge $e{:=}\{u{,}v\}$. Each corner vertex of H_e corresponds to a vertex of e.

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. 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 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 $H{:=}f\,G$. 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 a vertex 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 vertices 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 vertex 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) Proof?.

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.

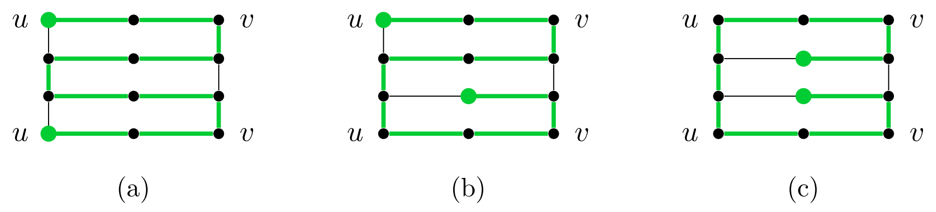

Figure 2: There are three different types of Hamiltonian paths for a subgraph H_e for an edge $e{:=}\{u,v\}$. Each corner vertex is labeled with its corresponding vertex of the edge e.

(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 vertices v of the graph 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 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 vertex of the edge e. This vertex 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.

6.1 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 assumed linear order on the vertices is used to identify different instances of the same undirected edge.

$$\begin{tikzcd}
(e,u,1) \arrow[r, dash] \arrow[d, dash] & (e,u,3) \arrow[r, dash] & (e,v,2) \arrow[d, dash] \\
(e,u,4) \arrow[r, dash] \arrow[d, dash] & (e,v,6) \arrow[r, dash] & (e,u,5) \arrow[d, dash] \\
(e,v,5) \arrow[r, dash] \arrow[d, dash] & (e,u,6) \arrow[r, dash] & (e,v,4) \arrow[d, dash] \\
(e,u,2) \arrow[r, dash] & (e,v,3) \arrow[r, dash] & (e,v,1)
\end{tikzcd}$$

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 instantiations of the functions f and g. This formalization approach is adapted from [2], who formalized the Depth-first search with this approach.

The vertices of the subgraphs H_e are represented with a triple (e,w,i) where w is a vertex 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. $H_{\{u,v\}}{=}H_{\{v,u\}}$ because $\{u,v\}$ is an undirected edge.

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 an assumed function that is used to identify different instances of undirected edges.

Explain functions...

```
$is\text{-}edge\text{-}in\text{-}He\ G\ (uEdge\ x\ y) =$
$fold\text{-}g1\text{-}uedges2\ (\lambda e\ b.\ b \vee isin2\ (\mathcal{N}_2\ (H_e\ e)\ x)\ y)\ G\ False$
```

```
$are\text{-}vertices\text{-}in\text{-}He\ G\ x\ y =$
$fold\text{-}g1\text{-}uedges2\ (\lambda e\ b.\ b \vee isin2\ (V_{He}\ e)\ x \wedge isin2\ (V_{He}\ e)\ y)\ G\ False$
```

```
$c\ G\ (e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2) =$
$($if $is\text{-}edge\text{-}in\text{-}He\ G\ (uEdge\ (e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2))$ then $1$
 else if $are\text{-}vertices\text{-}in\text{-}He\ G\ (e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2)$
      then $the\text{-}enat\ (min\text{-}dist\text{-}in\text{-}He\ G\ (e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2))$
      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 tour contains a Hamiltonian path for every subgraph H_e. The function *vc-of-tour* computes a vertex cover for G from a given Hamiltonian cycle that contains a Hamiltonian path for every subgraph H_e.

```
$hp\text{-}starting\text{-}at\ G\ ((e,\ w,\ i) \cdot T) =$
$($case $rep1\ e$ of
 $uEdge\ u\ v \Rightarrow$
   if $to\text{-}covering\text{-}vertex\ G\ ((e,\ w,\ i) \cdot T) = u$ then $hp\text{-}u1\ e$ else $hp\text{-}v1\ e)$
$hp\text{-}starting\text{-}at\ G\ [] = undefined$
```

```
$replace\text{-}hp\ G\ [] = []$
$replace\text{-}hp\ G\ (x \cdot T) =$
$hp\text{-}starting\text{-}at\ G\ (x \cdot T)\ @$
$replace\text{-}hp\ G\ (filter\ (\lambda y.\ \neg\ are\text{-}vertices\text{-}in\text{-}He\ G\ x\ y)\ T)$
```

```
$shorten\text{-}tour\ G\ T =$
$($case $rotate\text{-}tour\ (\lambda(e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2).\ rep1\ e_1 \neq rep1\ e_2)\ T$ of
 $x \cdot y \cdot T' \Rightarrow last\ (hp\text{-}starting\text{-}at\ G\ (y \cdot T')) \cdot$
   $replace\text{-}hp\ G\ (filter\ (\lambda z.\ \neg\ are\text{-}vertices\text{-}in\text{-}He\ G\ y\ z)\ T')\ @$
   $hp\text{-}starting\text{-}at\ G\ (y \cdot T'))$
```

```
$vc\text{-}of\text{-}tour\ G\ [] = set\text{-}empty1$
$vc\text{-}of\text{-}tour\ G\ (x \cdot T) =$
$insert1\ (to\text{-}covering\text{-}vertex\ G\ (x \cdot T))$
 $(vc\text{-}of\text{-}tour\ G\ (filter\ (\lambda y.\ \neg\ are\text{-}vertices\text{-}in\text{-}He\ G\ x\ y)\ T))$
```

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

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 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 path 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 path 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 [12, Theorem 21.1].

Lemma 6.1. *is-hc* $(f\ G)\ T \Longrightarrow$
cost-of-path$_c$ (*shorten-tour* $G\ T$) $\leq$ *cost-of-path*$_c$ T

intuition for shorten-tour

6.2 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 the L-reduction.

The function f constructs a complete graph.

Theorem 6.2. *is-complete* $(f\ G)$

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

Theorem 6.3. *is-hc* $(f\ G)\ T \Longrightarrow$ *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.

I 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 [12, 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.

Lemma 6.4. *cost-of-path*$_c$ $OPT_H \leq 15 * |E(G)| + |OPT_{VC}|$

To prove this lemma, 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.

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

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.

Lemma 6.5. *is-vc* $G\ X \Longrightarrow$
$\exists\, T.$ *is-hc* $(f\ G)\ T \wedge$ *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 [12, Theorem 21.1].

Lemma 6.6. *is-hc* $(f\ G)\ T \Longrightarrow$
$\exists\, k \geq |g\ G\ T|.\ 15 * |E(G)| + k \leq$ *cost-of-path*$_c$ T

graphical representation?

Intuitively, this follows from the fact that 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 vertex 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 Lemma 6.4 we prove the first linear inequality required for the L-reduction. By assumption, the degree of every vertex in G is at most 4. From Lemma 3.4, we conclude that the number of edges of G is at most 4-times the cardinality of optimal vertex cover OPT_{VC} of G. The following chain of inequalities derives the first inequality required for the L-reduction.

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

The following theorem states the first inequality required for the L-reduction.

Theorem 6.7. $\textit{cost-of-path}_c\ OPT_H \leq \mathit{61} * |OPT_{VC}|$

The second inequality for the L-reduction follows from Lemma 6.4 and Lemma 6.6. The following chain of inequalities derives the second linear inequality for the L-reduction.

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

The following theorem states the second inequality required for the L-reduction.

Theorem 6.8. $\textit{is-hc}\ (f\ G)\ T$
$\Longrightarrow ||g\ G\ T| - |OPT_{VC}|| \leq \mathit{1} * |\textit{cost-of-path}_c\ T - \textit{cost-of-path}_c\ OPT_H|$

6.3 Incompleteness and Unfinished Business

In this section, I point out two cases that are not covered in the L-reduction proof by [12, Theorem 21.1]. I also describe the parts of my formalization that are not finished.

The L-reduction proof that is presented in [12, Theorem 21.1] is incomplete and does not cover the following cases.

(i) The proof by [12, 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 (Lemma 6.4). 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 [12, 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 a subgraph H_e that do not start or end at corner vertices of the subgraph H_e (see Figures 2). In [12], 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 [12, 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 [12]. On the other hand, I defined the function g using the function *shorten-tour* which explicitly covers these cases.

    The proof in [12, Theorem 21.1] references a proof by [16] that 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 [16] 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 a 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 a subgraph H_e. My goal was to use this abstract lemma, 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.

7 Future Work and Conclusion

In this work, I present my formalization of parts of section 21.1, *Approximation Algorithms for the TSP* from [12] with the interactive theorem prover Isabelle/HOL. I continue the work of [8] by formalizing and formally verifying two approximation algorithms for the Metric TSP: the DoubleTree and the Christofides-Serdyukov algorithm. For that, I build on the formalization of graphs by Abdulaziz [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 the 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 $\mathrm{P} = \mathrm{NP}$.

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, formalizations of algorithms to compute a MST, a minimum perfect matching, and an Eulerian tour are needed. I have started (but not finished) to implement an adaptor between my formalization and the formalization of Prim's algorithm in Isabelle/HOL by Lammich and Nipkow [13]. Suitable algorithms for the other problems which would need to be formalized are e.g. Edmonds' Blossom algorithm [7, 6] for minimum perfect matching and Euler's algorithm [12] for Eulerian tour.

check min match citation!

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-` [21]. The running times could then be proven with the refined functions.

References

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

[2] Mohammad Abdulaziz. Formalization Depth-first search algorithm. GitHub Repository, 2022. https://github.com/cmadlener/archive-of-graph-formalizations/blob/master/Pair_Graph/DFS.thy.

[3] Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences of the United States of America*, 43(9): 842–844, 1957. ISSN 00278424.

[4] B.L. 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, 1996.

[5] Nicos 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.

[6] Jack Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics*, pages 125–130, 1965.

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

[8] Robin Eßmann, Tobias Nipkow, and Simon Robillard. Verified approximation algorithms. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning*, pages 291–306, Cham, 2020. Springer International Publishing.

[9] 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.

[10] Vojtch Jarník. About a certain minimal problem (translated from czech). *Práce Moravské Pírodovdecké Spolenosti (in Czech)*, 6:57–63, 1930.

[11] Anna R. Karlin, Nathan Klein, and Shayan Oveis 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. ISBN 9781450380539.

[12] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Berlin, Heidelberg, 6th edition, 2018. ISBN 978-3-662-56039-6.

[13] Peter Lammich and Tobias Nipkow. Purely functional, simple, and efficient implementation of Prim and Dijkstra. *Archive of Formal Proofs*, June 2019. ISSN 2150-914x. https://isa-afp.org/entries/Prim_Dijkstra_Simple.html, Formal proof development.

[14] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Lecture Notes in Computer Science. Springer, 2002.

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

[16] Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, Feb 1993.

[17] R. C. Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.

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

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

[20] Vijay V. Vazirani. *Approximation Algorithms*. Springer Berlin, Heidelberg, 2013. ISBN 3642084699.

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