Formalizing Theory of Approximating the Traveling-Salesman Problem

Lukas Koller

May 20, 2023

Abstract

The Traveling Salesman Problem (TSP) is one of the most well-known NP-complete optimization problems. There is no constant-factor approximation algorithm for TSP. The Metric TSP is a simlification of the TSP where it is assumed that the input graph is complete and the edge weights satisfy the triangle-inequality. This work is split into two parts. The first part describes the formalization and formal verification of two approximation algorithm for the Metric TSP: the DoubleTree and Christofides-Serdyukov algorithm. The Christofides-Serdyukov is the best known approxmiation algorithm for Metric TSP with an approximation ratio of $\frac{3}{2}$. The second part 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 **2**

2 Related Work **3**

3 Fundamentals and Definitions **3**

3.1 Connected Graphs 4
3.2 Weighted Graphs 4
3.3 Complete Graphs 4
3.4 Acyclic Graphs 5
3.5 Trees 5
3.6 Hamiltonian Cycles 6
3.7 Traveling-Salesman Problem 6
3.8 Multi-Graphs 6
3.9 Eulerian Tours 6
3.10 Perfect Matchings 7

4 Approximating the Metric TSP **7**
4.1 Formalizing the DoubleTree Algorithm 8
4.2 Formalizing the Christofides-Serdyukov Algorithm . . . 10

5 L-Reduction from VCP4 to Metric TSP **12**

6 Future Work and Conclusion **17**

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 graph s.t. every vertex of the graph is visited exactly once. The TSP has many applications in different areas, e.g. planning and scheduling [1], logistics [10], designing printed circuit boards [5], etc..

The TSP is NP-hard [6]. This means, that unless P = NP, the TSP cannot be solved in polynomial time. When dealing with optimization problems, a typical strategy is to approximate the optimal solution with an approximation algorithm. An approximation algorithm is a polynomial time algorithm that returns a bounded approximate solution. An approximation algorithm essentially trades the optimality of the returned solution for polynomial running time. Ideally, the distance between the approximate solution and the optimal solution can be bounded by a constant factor. Unfortunately, there is no constant-factor approximation algorithm for the TSP [6].

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 [6].

This report describes the formalization of parts of the section 21.1, *Approximation Algorithms for the TSP* from [6].

1. I have formalized and verified two approximation algorithms, DoubleTree and Christofides-Serdyukov, for Metric TSP. The Christofides-Serdyukov [2, 11] algorithm is the best known approximation algorithm for the Metric TSP [6].

2. I have formalized a L-reduction from the Minimum Vertex Cover Problem with maximum degree of 4 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 [8]. An approximation scheme is an approximation algorithm that takes an additional parameter $\epsilon > 0$ based on which the approximate solution is bounded. MaxSNP is a complexity class that contains

graph-theoretical optimization problems. For any problem in MaxSNP there is a constant-factor approximation algorithm [8]. A linear reduction (L-reduction) is special type of reduction for optimization problems [8]. A problem is called MaxSNP-hard if every problem in MaxSNP L-reduces to it.

2 Related Work

[4] have formalized different approximation algorithms in Isabelle/HOL.
Reductions in Isabelle?
Refinement to IMP-.

3 Fundamentals and Definitions

I build on the exiting formalization of Berge's theorem by Mohammad Abdulaziz (citation?). 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 Mohammad Abdulaziz (citation?) 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 the formalization of Berge's theorem by Mohammad Abdulaziz (citation?), e.g. weighted edges, minimum spanning-tree, Hamiltonian cycle, etc..

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, thus 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.

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

3.2 Weighted Graphs

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

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

3.3 Complete Graphs

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

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

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.

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

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.

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

3.4 Acyclic Graphs

A path is a simple if no edge is used twice.

is-simple P ≡ *distinct* (*edges-of-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-cycle E C ≡ (∃ *v*. *walk-betw E v C v*) ∧ *is-simple C* ∧ *0* < |*edges-of-path C*|

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

is-acyclic E ≡ ∄ *C*. *is-cycle E C*

A vertex that is contained in a cycle has a degree of at least 2.

is-cycle E C ∧ *v* ∈ *C* ⟹ *2* ≤ *degree E v*

3.5 Trees

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

is-tree T ≡ *is-connected T* ∧ *is-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-st E T ≡ *T* ⊆ *E* ∧ *Vs E* = *Vs T* ∧ *is-tree T*

A minimum spanning tree (MST) is a spanning tree with minimum total weight. The function *cost-of-st*$_c$ computes the total weight of a given spanning tree with the edge weights *c*.

is-mst E c T ≡ *is-st E T* ∧ (∀ *T′*. *is-st E T′* ⟶ *cost-of-st*$_c$ *T* ≤ *cost-of-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-perf-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-min-match E c M $\equiv$
is-perf-match E M $\wedge$
$(\forall M'.$ *is-perf-match E M'* $\longrightarrow$ *cost-of-match*$_c$ *M* $\leq$ *cost-of-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.

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 (Lemma 21.3, [6]):

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 (Lemma 21.3 [6]) 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.

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 [7], and

2. an algorithm to compute a Eulerian tour in an Eulerian multigraph, e.g. Eulers's algorithm [6].

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

is-hc E double-tree

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

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

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

VARS T T_{2x} v P P' H
 { *True* }
  T := *comp-mst c E*;
  T_{2x} := *mset-set* T + *mset-set* T;
  P := *comp-et* T_{2x};
  P' := P;
  H := [];
  WHILE $P' \neq$ []
  INV { *comp-hc-of-et* P [] = *comp-hc-of-et* P' H
    $\wedge$ P = *comp-et* T_{2x}
    $\wedge$ T_{2x} = *mset-set* T + *mset-set* T
    $\wedge$ T = *comp-mst c E*
  } *DO*

```
    v := hd P';
    P' := tl P';
    IF v ∈ set H ∧ P' ≠ [] THEN
      H := H
    ELSE
      H := v#H
    FI
  OD
{ is-hc E H ∧ cost-of-path_c H ≤ 2 * cost-of-path_c OPT }
```

4.2 Formalizing the Christofides-Serdyukov Algorithm

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

1. Compute a MST T of the input graph E.
2. Compute a minimum perfect-matching M between the vertices that have odd degree in T.
3. Compute a Eulerian tour P of the union of the MST and the perfect-matching $J = T + M$.
4. Remove duplicate vertices in P by short-cutting (see *comp-hc-of-et*).

Hence, the Christofides-Serdyukov algorithm depends on 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 [3].

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.

$(2 :: 'b) * \textit{cost-of-path}_c\ \textit{christofides-serdyukov}$
$\leq (3 :: 'b) * \textit{cost-of-path}_c\ OPT$

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

```
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' ≠ []
   INV { comp-hc-of-et P [] = comp-hc-of-et P' H
     ∧ P = comp-et J
     ∧ J = mset-set T + mset-set M
     ∧ M = comp-match ({e ∈ E. e ⊆ W}) c
     ∧ W = {v ∈ Vs T. ¬ even' (degree T v)}
     ∧ T = comp-mst c E
   } DO
     v := hd P';
     P' := tl P';
     IF v ∈ set H ∧ P' ≠ [] THEN
       H := H
     ELSE
       H := v#H
     FI
   OD
 { is-hc E H ∧ 2 * cost-of-path_c H ≤ 3 * cost-of-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 that is presented in [6] (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 every vertex has a degree of at most 4. The bounded degree is only needed to prove the linear inequalities required by the L-reduction.

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

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.

We describe the function f with the following construction. Let G be an instance of the VCP4, and we denote the constructed graph with $H{:=}f\ G$. For each edge e of G we add a subgraph H_e to H (see figure 6). The graph H is the complete graph over all the vertices of 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 each subgraph H_e corresponds to an endpoint of the edge e.

Each 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. Therefore, the start- and end-vertex

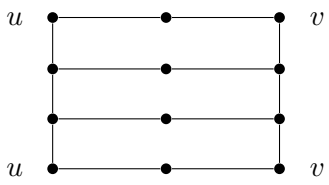

Figure 1: Subgraph H_e for an edge $e{:=}\{u,v\}$. For each corner vertex the corresponding endpoint is labeled.

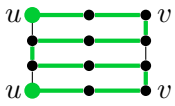

Figure 2

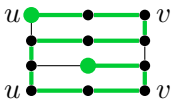

Figure 3

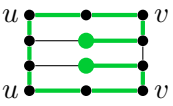

Figure 4

Figure 5: This figure shows the different types Hamiltonian paths for the subgraphs H_e for an edge $e{:=}\{u,v\}$.

of a Hamiltonian path for a subgraph H_e can only correspond to the same 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 5).

Next, we describe the edge weights of the graph H. The graph H is complete, thus we have edges between every pair of vertices of H. Every vertex of H belongs to a subgraph H_e. We distinguish 3 types of edges to describe the edge weights: (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 is omitted.

Next, we 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 we identify the covering vertex of e by looking at the Hamiltonian path of 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, we construct a Hamiltonian cycle T' for H s.t. T' contains a Hamiltonian path for every H_e and the cost of T' is at most the cost of T. The Hamiltonian cycle T' is constructed by

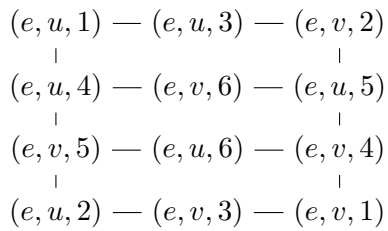

Figure 6: Formalization of the subgraph H_e for an edge $e{:=}\{u,v\}$.

iteratively replacing parts of the Hamiltonian cycle T with a Hamiltonian path for a subgraph H_e.

The functions f and g have to be computable (in polynomial time) functions. Therefore, the locale *ugraph-adj-map* provides an abstract adjacency map, which can be instantiated to obtain executable functions on graphs.

The L-reduction itself is formalized in the locale *VC4-To-mTSP* that depends on two executable adjacency-maps (*ugraph-adj-map*), one for each of the graphs G and H (denoted by *g1* and *g2*). The locale *VC4-To-mTSP* also assumes appropriate `fold`-functions for the graphs. The necessary graph-related definitions are redefined for the adjencecy-map representation of graphs. The definitions for the adjencecy-map representation of graphs are denoted by the postfix $-Adj$.

The reduction functions f and g are defined in the locale *VC4-To-mTSP* using the assumed `fold`-functions and some auxiliary functions. The function f uses the auxiliary function *complete-graph* to construct the complete graph for a given set of vertices. The function V_H computes all the vertices of the graph H. 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 naming of the vertices of the subgraph H_e has to be symmetric s.t. $H_{\{u,v\}}{=}H_{\{v,u\}}$. I attempted to encode more information with the naming, but I failed.

$V_H\ G =$ *fold-g1-uedges1* $(union2 \circ V_{He})\ G$ *set-empty2*

$f\ G =$ *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 identify the different cases. To simplify things, the case for two vertices that are neighbours in subgraph H_e (weight 1). 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.

```
c G (e1, w1, i1) (e2, w2, i2) =
(if is-edge-in-He G (uEdge (e1, w1, i1) (e2, w2, i2)) then 1
 else if are-vertices-in-He G (e1, w1, i1) (e2, w2, i2)
      then the-enat (min-dist-in-He G (e1, w1, i1) (e2, w2, i2))
      else if rep1 e1 ≠ rep1 e2 ∧
              w1 = w2 ∧ (i1 = 1 ∨ i1 = 2) ∧ (i2 = 1 ∨ i2 = 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 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-of-tour G (tl (shorten-tour G T))
```

Given a tour T of H, the function *shorten-tour* iteratively (for each edge e of G) removes all vertices of the subgraph H_e from the tour T and 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. The appended Hamiltonian path for the subgraph H_e is carefully chosen such that the resulting tour has less total weight. On the other hand, the function g descibed by [6] only inserts a Hamiltonian paths when the tour T does not already contain a Hamiltonian path for the subgraph H_e.

The main reason for this decision is that, by always inserting a Hamiltonian path we make sure that the resulting tour only contains Hamiltonian paths that start and end at a corner vertex of a subgraph H_e. This simplifies the formalization. The resulting tour is represented with a sequence of starting-vertices instead of a path in the graph H. This simplifies the definitions and proofs, e.g. definition for *vc-of-tour*. Furthermore, all the cases for the different possible Hamiltonian paths for the subgraphs H_e (see figure 5) are handled explicitly. The cases where the tour T contains a Hamiltonian path for a subgraph H_e of type 3 or 4 from figure 5 are not covered by the proof of [6].

The locale *VC4-To-mTSP* also contains the proofs for the feasibility of the reduction functions and the linear inequalities required for a L-reduction.

```
g1.ugraph-adj-map-invar G ⟹ g2.is-complete-Adj (f G)
```

```
g1.ugraph-adj-map-invar G ∧ 1 < |g1.uedges G| ∧ g2.is-hc-Adj (f G) T ⟹
g1.is-vc-Adj G (g G T)
```

$[\![g1.ugraph\text{-}adj\text{-}map\text{-}invar\ G;\ 1 < |g1.uedges\ G|;\ 1 < card1\ OPT_{VC};$
$\bigwedge v.\ v \in g1.vertices\ G \Longrightarrow enat\ (g1.degree\text{-}Adj\ G\ v) \leq enat\ 4;$
$g1.is\text{-}min\text{-}vc\text{-}Adj\ G\ OPT_{VC} \wedge$
$set\text{-}invar1\ OPT_{VC} \wedge g2.is\text{-}tsp\text{-}Adj\ (f\ G)\ (c\ G)\ OPT_{mTSP}]\!]$
$\Longrightarrow cost\text{-}of\text{-}path\ (c\ G)\ OPT_{mTSP} \leq 61 * card1\ OPT_{VC}$

$g1.ugraph\text{-}adj\text{-}map\text{-}invar\ G \wedge$
$1 < |g1.uedges\ G| \wedge$
$g1.is\text{-}min\text{-}vc\text{-}Adj\ G\ OPT_{VC} \wedge$
$set\text{-}invar1\ OPT_{VC} \wedge$
$1 < card1\ OPT_{VC} \wedge$
$g2.is\text{-}tsp\text{-}Adj\ (f\ G)\ (c\ G)\ OPT_{mTSP} \wedge g2.is\text{-}hc\text{-}Adj\ (f\ G)\ T \Longrightarrow$
$|card1\ (g\ G\ T) - card1\ OPT_{VC}|$
$\leq 1 * |cost\text{-}of\text{-}path\ (c\ G)\ T - cost\text{-}of\text{-}path\ (c\ G)\ OPT_{mTSP}|$

proof intuition

The following lemmas are important lemmas when proving the linear inequalities.

$g1.ugraph\text{-}adj\text{-}map\text{-}invar\ G \wedge$
$1 < |g1.uedges\ G| \wedge$
$1 < card1\ OPT_{VC} \wedge$
$g1.is\text{-}min\text{-}vc\text{-}Adj\ G\ OPT_{VC} \wedge$
$set\text{-}invar1\ OPT_{VC} \wedge$
$g2.is\text{-}tsp\text{-}Adj\ (f\ G)\ (c\ G)\ OPT_{mTSP} \wedge 1 < |g1.uedges\ G| \wedge 1 < card1\ OPT_{VC}$
$\Longrightarrow$
$cost\text{-}of\text{-}path\ (c\ G)\ OPT_{mTSP} \leq 15 * |g1.uedges\ G| + card1\ OPT_{VC}$

$[\![g1.ugraph\text{-}adj\text{-}map\text{-}invar\ G;\ 1 < |g1.uedges\ G|;\ g1.is\text{-}min\text{-}vc\text{-}Adj\ G\ OPT_{VC};$
$set\text{-}invar1\ OPT_{VC};\ 1 < card1\ OPT_{VC};\ g2.is\text{-}hc\text{-}Adj\ (f\ G)\ T;$
$\bigwedge k.\ card1\ (g\ G\ T) \leq k \wedge 15 * |g1.uedges\ G| + k \leq cost\text{-}of\text{-}path\ (c\ G)\ T \Longrightarrow$
$thesis]\!]$
$\Longrightarrow thesis$

The following lemma describes a case distinction on the different possible Hamiltonian paths that are admitted by the substructures H_e.

$[\![g1.ugraph\text{-}adj\text{-}map\text{-}invar\ G;\ e \in g1.uedges\ G;\ rep1\ e = uEdge\ u\ v;$
$xs = g2.vertices\ (H_e\ e);\ distinct\ xs;\ cost\text{-}of\text{-}path\ (c\ G)\ xs = 11;$
$hd\ xs = (e,\ u,\ 1) \wedge last\ xs = (e,\ u,\ 2) \Longrightarrow thesis;$
$hd\ xs = (e,\ u,\ 1) \wedge last\ xs = (e,\ u,\ 6) \Longrightarrow thesis;$
$hd\ xs = (e,\ u,\ 2) \wedge last\ xs = (e,\ u,\ 1) \Longrightarrow thesis;$
$hd\ xs = (e,\ u,\ 2) \wedge last\ xs = (e,\ v,\ 6) \Longrightarrow thesis;$
$hd\ xs = (e,\ u,\ 6) \wedge last\ xs = (e,\ u,\ 1) \Longrightarrow thesis;$
$hd\ xs = (e,\ u,\ 6) \wedge last\ xs = (e,\ v,\ 2) \Longrightarrow thesis;$
$hd\ xs = (e,\ u,\ 6) \wedge last\ xs = (e,\ v,\ 6) \Longrightarrow thesis;$
$hd\ xs = (e,\ v,\ 1) \wedge last\ xs = (e,\ v,\ 2) \Longrightarrow thesis;$
$hd\ xs = (e,\ v,\ 1) \wedge last\ xs = (e,\ v,\ 6) \Longrightarrow thesis;$

$hd\ xs = (e,\ v,\ 2) \wedge last\ xs = (e,\ v,\ 1) \Longrightarrow thesis;$
$hd\ xs = (e,\ v,\ 2) \wedge last\ xs = (e,\ u,\ 6) \Longrightarrow thesis;$
$hd\ xs = (e,\ v,\ 6) \wedge last\ xs = (e,\ v,\ 1) \Longrightarrow thesis;$
$hd\ xs = (e,\ v,\ 6) \wedge last\ xs = (e,\ u,\ 2) \Longrightarrow thesis;$
$hd\ xs = (e,\ v,\ 6) \wedge last\ xs = (e,\ u,\ 6) \Longrightarrow thesis\rrbracket$
$\Longrightarrow thesis$

This lemma is not proven in my formalization because I am unsure about the best way to prove this lemma. In the theory `./reductions/FindHamiltonianPath.thy` I have started to prove this lemma for an instantiation of the graph representations. This result needs to be transfered into the reduction proof. At the moment I am not sure what the best and way to do this is.

The L-reduction proof that is presented in [6] is incomplete.

(i) The proof fails if the optimal vertex cover of G has a cardinality of 1. In this case, some of the auxiliary lemmas used in [6] do not hold. The construction of the L-reduction still works in this case, but for the proof this case needs to be considered separately. For now I have only excluded this case through assumptions.

(ii) The proof is missing multiple cases, when identifying the covering vertices. There are different Hamiltonian paths for the substructures H_e that do not start or end at corners of the substructures H_e. [6] only describe how to identify a covering vertex if the Hamiltonian path starts and ends at a corner of the substructures H_e. For some Hamiltonian path that do not start or end at corners of the substructures 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. compare to [9]

6 Future Work and Conclusion

For my Interdisciplinary Project, I have formalized parts of the section 21.1, *Approximation Algorithms for the TSP* from [6].

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

Future work includes:

- The Eulerian graphs that are constructed for the DoubleTree and Christofides-Serdyukov algorithm might be multi-graphs. In the theory `./graphs/MultiGraph.thy`, I have started formalizing an encoding of multi-graphs with simple graphs. This formalization needs to be finished.
- Prove the existence of the necessary algorithms for the DoubleTree and Christofides-Serdyukov algorithm.
  - Write an adapter to the existing formalization of Prim's algorithm [7].
  - Formalize and verify algorithms for minimum perfect matching [3] and Eulerian tour [6].
- Prove the polynomial running time of reduction functions f and g.
  - To prove the running time one needs to assume a computational model. `IMP-` provides a computational model. Thus, a way to prove the polynomial time-complexity of the functions f and g it to refine the functions to `IMP-` and prove the running time of the refined definitions.

References

[1] 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.

[2] 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.

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

[4] 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.

[5] 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.

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

[7] 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.

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

[9] C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18(1):111, feb 1993.

[10] 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.

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