Formalizing Theory of Approximating the Traveling-Salesman Problem

Lukas Koller

May 13, 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

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 [9], 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, 10] 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, where each 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 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..

In this section, I will go 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. Because most of the definitions are straightforward I will only highlight specific important formalization choices.

All of these definitions are in `.thy`-files, which are located in the `./graphs`- or `./problems`-directory.

Many simple results and lemmas are formalized in the theory *tsp.Misc* which is located in the directory `./misc`.

3.1 Connected Graphs

A graph is connected if any pair of vertices is 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* assumes the edge weights c to be positive.

3.3 Complete Graphs

A graph is complete if there exists an edge for any pair of vertices that are not equal.

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

The locale *compl-graph-abs* fixes an instance of a complete graph E.
In a complete graph, any sequence of vertices s.t. no two consecutive vertices are equal is a path.

$\textit{compl-graph-abs}\ E \wedge \textit{distinct-adj}\ P \wedge P \subseteq \textit{Vs}\ E \Longrightarrow \textit{path}\ E\ P$

The locale *restr-compl-graph-abs* additionally fixes a subgraph of the instance E.
The locale *metric-graph-abs* fixes an instance of a complete graph E and assumes that the edge weights c are positive and satisfy the triangle-inequality.
A consequence of these assumption is that any path can be short-cutted.

$\textit{metric-graph-abs}\ E\ c \wedge P \subseteq \textit{Vs}\ E \Longrightarrow$
$\textit{cost-of-path}\ (\lambda u\ v.\ c\ \{u,\ v\})\ (\textit{short-cut}\ E_V\ P)$
$\leq \textit{cost-of-path}\ (\lambda u\ v.\ c\ \{u,\ v\})\ P$

3.4 Acyclic Graphs

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

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

A cycle in a graph is a vertex-path where the first and last element are the same.
A cycle is a vertex-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.

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

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

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

A vertex that is contained in a cycle has a degree greater-equal to 2.

$\textit{graph-abs}\ E \wedge \textit{is-cycle}\ E\ C \wedge v \in C \Longrightarrow \mathit{2} \leq \textit{degree}\ E\ v$

3.5 Trees

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

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

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

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

A minimum spanning tree is a spanning tree with minimum total weight.

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

The locale *mst* assumes the existence of an algorithm to compute a minimum spanning tree.

3.6 Hamiltonian Cycles

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

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

With this formalization, a Hamiltonian cycle is not necessarily a cycle. If a graph only contains one edge $e=\{u,v\}$ then $[u,v,u]$ is a Hamiltonian cycle but not a cycle because it is not a simple path. The edge e is used twice.

3.7 Traveling-Salesman Problem

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

$\textit{is-tsp}\ E\ c\ P \equiv$
$\textit{is-hc}\ E\ P \wedge (\forall P'.\ \textit{is-hc}\ E\ P' \longrightarrow \textit{cost-of-path}\ c\ P \leq \textit{cost-of-path}\ c\ P')$

3.8 Multi-Graphs

I have started to formalize an encoding of a multi-graph with a regular graph. My attempts be found in the theory *tsp.MultiGraph*.

3.9 Eulerian Tours

A graph is eulerian if every vertex has even degree.

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

A Eulerian tour is a path that uses every edge of a given graph exactly once.

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

The locale *eulerian* fixes an algorithm to compute a Eulerian tour for a given multi-graph.

3.10 Perfect Matchings

The existing graph formalization by Mohammad Abdulaziz already contains formalization of matchings. A perfect matching is a matching that contains every vertex.

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

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

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

The locale *min-weight-matching* fixes an algorithm 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 G with edge weights c be an instance of the Metric TSP. Given a connected Eulerian graph G' that spans the vertices of the graph G, we can compute (in polynomial time) a Hamiltonian cycle for G with total weight of at most the total of all edges in G'.

The proof for this proposition is constructive: first a Eulerian tour P is computed in the graph G'. The tour P visits every vertex of G at least once, because the Eulerian graph G' spans the vertices of G. For the next step, duplicate vertices in the tour P are removed to obtain the desired

Hamiltonian cycle for the graph G ("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 the total weight of the edges of the Eulerian graph G'.

This construction for this proposition is formalized by the function *comp-hc-of-et*. The locale *hc-of-et* provides the necessary assumption on the input graph G. The following lemmas prove the correctness.

hc-of-et E c comp-et $\wedge$ *is-et X P* $\wedge$ *mVs X* $=$ *Vs E* $\wedge$ *set-mset X* $\subseteq$ *E* $\Longrightarrow$
is-hc E (*comp-hc-of-et P* [])

hc-of-et E c comp-et $\wedge$ *P* $\cup$ *H* $\subseteq$ *Vs E* $\Longrightarrow$
cost-of-path ($\lambda u\ v.\ c\ \{u,\ v\}$) (*comp-hc-of-et P H*)
$\leq$ *cost-of-path* ($\lambda u\ v.\ c\ \{u,\ v\}$) (*rev P* @ *H*)

Using the proposition (Lemma 21.3 [6]), an approximation algorithm for the Metric TSP is derived by simply constructing a Eulerian graph that spans the vertices of the graph G. By minimizing the total weight of the edges of the Eulerian graph one directly reduces the approximation ratio of the resulting approximation algorithm.

Both approximation algorithms, DoubleTree and Christofides-Serdyukov, first compute a minimum spanning-tree T of G and add edges to T to construct a Eulerian graph that spans the vertices of the graph G. By the proposition (Lemma 21.3 [6]), we can then compute (in polynomial time) a Hamiltonian cycle in G with a total weight that is bounded by the total weight of T and the added edges.

The DoubleTree algorithm constructs a Eulerian graph by computing the multi-graph which contains every edge of T twice. 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 bounded by 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 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 1.5-approximation algorithm for the Metric TSP.

4.1 Formalizing the DoubleTree Approximation Algorithm

The DoubleTree algorithm consists of three main steps:

1. compute a minimum spanning-tree T of the input graph G,
2. compute a Eulerian tour P of the doubled minimum spanning-tree $T + T$,
3. and remove duplicate vertices in P by short-cutting.

Thus, the DoubleTree algorithm depends to two algorithms:

1. an algorithm to compute a minimum spanning-tree, e.g. Prim's algorithm [7], and
2. an algorithm to compute a Eulerian tour in an Eulerian graph, 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 assumptions are both captured in the locale *hc-of-et*. Further, to simplify matters the locale fixes a graph E with edge weights c and add the assumptions for an instance of Metric TSP (*metric-graph-abs*). Defining the DoubleTree algorithm in the locale *double-tree-algo* is straightforward.

$$\begin{array}{l} \textit{double-tree} = \\ (\textsf{let}\ T = \textit{comp-mst}\ c\ E;\ T_{2x} = \textit{mset-set}\ T + \textit{mset-set}\ T;\ P = \textit{comp-et}\ T_{2x} \\ \ \textsf{in}\ \textit{comp-hc-of-et}\ P\ []) \end{array}$$

For the defined DoubleTree algorithm we prove two properties: (i) the feasibility, and (ii) the approximation factor. The formalization of the proofs for the feasibility and approximation ratio of the DoubleTree algorithm is straightforward. The following lemmas prove the feasibility and the approximation ratio of the DoubleTree algorithm.

$$\begin{array}{l} [\![\textit{graph-invar}\ E;\ \textit{is-complete}\ E;\ \bigwedge e.\ (0 :: 'b) < c\ e; \\ \ \bigwedge u\ v\ w.\ u \in \textit{Vs}\ E \wedge v \in \textit{Vs}\ E \wedge w \in \textit{Vs}\ E \Longrightarrow c\ \{u, w\} \leq c\ \{u, v\} + c\ \{v, w\}; \\ \ \bigwedge E.\ \textit{is-connected}\ E \Longrightarrow \textit{is-mst}\ E\ c\ (\textit{comp-mst}\ c\ E); \\ \ \bigwedge E.\ \textit{is-eulerian}\ E \Longrightarrow \textit{is-et}\ E\ (\textit{comp-et}\ E)]\!] \\ \Longrightarrow \textit{is-hc}\ E\ (\textit{double-tree-algo.double-tree}\ E\ c\ \textit{comp-et}\ \textit{comp-mst}) \end{array}$$

$$\begin{array}{l} [\![\textit{graph-invar}\ E;\ \textit{is-complete}\ E;\ \bigwedge e.\ (0 :: 'b) < c\ e; \\ \ \bigwedge u\ v\ w.\ u \in \textit{Vs}\ E \wedge v \in \textit{Vs}\ E \wedge w \in \textit{Vs}\ E \Longrightarrow c\ \{u, w\} \leq c\ \{u, v\} + c\ \{v, w\}; \\ \ \textit{is-tsp}\ E\ (\lambda u\ v.\ c\ \{u, v\})\ \textit{OPT}; \\ \ \bigwedge E.\ \textit{is-connected}\ E \Longrightarrow \textit{is-mst}\ E\ c\ (\textit{comp-mst}\ c\ E); \\ \ \bigwedge E.\ \textit{is-eulerian}\ E \Longrightarrow \textit{is-et}\ E\ (\textit{comp-et}\ E)]\!] \\ \Longrightarrow \textit{cost-of-path}\ (\lambda u\ v.\ c\ \{u, v\}) \\ \qquad (\textit{double-tree-algo.double-tree}\ E\ c\ \textit{comp-et}\ \textit{comp-mst}) \\ \qquad \leq (2 :: 'b) * \textit{cost-of-path}\ (\lambda u\ v.\ c\ \{u, v\})\ \textit{OPT} \end{array}$$

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

```
$[\![\textit{graph-invar } E;\ \textit{is-complete } E;\ \bigwedge e.\ (0 :: 'b) < c\ e;$
$\bigwedge u\ v\ w.\ u \in \textit{Vs } E \land v \in \textit{Vs } E \land w \in \textit{Vs } E \Longrightarrow c\ \{u,\ w\} \leq c\ \{u,\ v\} + c\ \{v,\ w\};$
$\textit{is-tsp } E\ (\lambda u\ v.\ c\ \{u,\ v\})\ \textit{OPT};$
$\bigwedge E.\ \textit{is-connected } E \Longrightarrow \textit{is-mst } E\ c\ (\textit{comp-mst } c\ E);$
$\bigwedge E.\ \textit{is-eulerian } E \Longrightarrow \textit{is-et } E\ (\textit{comp-et } E)]\!]$
$\Longrightarrow \{\textit{True}\}$
    $T := \textit{comp-mst } c\ E;$
    $T_{2x} := \textit{mset-set } T + \textit{mset-set } T;$
    $P := \textit{comp-et } T_{2x};$
    $P' := P;$
    $H := [];$
    $\textit{WHILE } P' \neq []$
    $\textit{INV } \{\textit{comp-hc-of-et } P\ [] = \textit{comp-hc-of-et } P'\ H \land$
         $P = \textit{comp-et } T_{2x} \land T_{2x} = \textit{mset-set } T + \textit{mset-set } T \land T = \textit{comp-mst } c\ E\}$
    $\textit{VAR } \{0\}$
    $\textit{DO } v := \textit{hd } P';$
       $P' := \textit{tl } P';\ \textit{IF } v \in H \land P' \neq []\ \textit{THEN SKIP ELSE } H := v \cdot H\ \textit{FI}$
    $\textit{OD}$
    $\{\textit{is-hc } E\ H \land$
     $\textit{cost-of-path } (\lambda u\ v.\ c\ \{u,\ v\})\ H$
     $\leq (2 :: 'b) * \textit{cost-of-path } (\lambda u\ v.\ c\ \{u,\ v\})\ \textit{OPT}\}$
```

4.2 Formalizing the Christofides-Serdyukov Approximation Algorithm

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

1. compute a minimum spanning-tree T of the input graph G,
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 minimum spanning-tree and the perfect-matching $T + M$,
4. and remove duplicate vertices in P by short-cutting.

Therefore, the Christofides-Serdyukov algorithm depends on the algorithm the DoubleTree algorithm depended on as well as an algorithm to compute a minimum perfect-matching, e.g. Edmond's Blossom algorithm [3] TODO: check if correct?.

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 definition of the CHRISTOFIDES-SERDYUKOV algorithm in the locale *christofides-serdyukov-algo* is straightforward.

$$\begin{aligned}
&\textit{christofides-serdyukov} = \\
&(\mathsf{let}\ T = \textit{comp-mst}\ c\ E;\ W = \{v \in \textit{Vs}\ T \mid \neg\ \textit{even}'\ (\textit{degree}\ T\ v)\}; \\
&\quad M = \textit{comp-match}\ \{e \in E \mid e \subseteq W\}\ c;\ J = \textit{mset-set}\ T + \textit{mset-set}\ M; \\
&\quad P = \textit{comp-et}\ J \\
&\ \mathsf{in}\ \textit{comp-hc-of-et}\ P\ [])
\end{aligned}$$

The feasibility and approximation ratio of the CHRISTOFIDES-SERDYUKOV algorithm are proven by the following lemmas. The formalization of their proofs is straightforward.

$$\textit{is-hc}\ E\ \textit{christofides-serdyukov}$$

$$\begin{aligned}
&(2 :: 'b) * \textit{cost-of-path}_c\ \textit{christofides-serdyukov} \\
&\le (3 :: 'b) * \textit{cost-of-path}_c\ \textit{OPT}
\end{aligned}$$

Like the DOUBLETREE algorithm, the definition of the CHRISTOFIDES-SERDYUKOV algorithm is refined to a `WHILE`-program using Hoare Logic.

```
$[\![\textit{graph-invar}\ E;\ \textit{is-complete}\ E;\ \bigwedge e.\ (0 :: 'b) < c\ e;$
 $\bigwedge u\ v\ w.\ u \in \textit{Vs}\ E \land v \in \textit{Vs}\ E \land w \in \textit{Vs}\ E \Longrightarrow c\ \{u, w\} \le c\ \{u, v\} + c\ \{v, w\};$
 $\textit{is-tsp}\ E\ (\lambda u\ v.\ c\ \{u, v\})\ \textit{OPT};$
 $\bigwedge E.\ \textit{is-connected}\ E \Longrightarrow \textit{is-mst}\ E\ c\ (\textit{comp-mst}\ c\ E);$
 $\bigwedge E.\ \textit{is-eulerian}\ E \Longrightarrow \textit{is-et}\ E\ (\textit{comp-et}\ E);$
 $\bigwedge E.\ \exists M.\ \textit{is-perf-match}\ E\ M \Longrightarrow \textit{is-min-match}\ E\ c\ (\textit{comp-match}\ E\ c)]\!]$
$\Longrightarrow \{\textit{True}\}$
    $T := \textit{comp-mst}\ c\ E;$
    $W := \{v \in \textit{Vs}\ T \mid \neg\ \textit{even}'\ (\textit{degree}\ T\ v)\};$
    $M := \textit{comp-match}\ \{e \in E \mid e \subseteq W\}\ c;$
    $J := \textit{mset-set}\ T + \textit{mset-set}\ M;$
    $P := \textit{comp-et}\ J;$
    $P' := P;$
    $H := [];$
    $\textit{WHILE}\ P' \neq []$
    $\textit{INV}\ \{\textit{comp-hc-of-et}\ P\ [] = \textit{comp-hc-of-et}\ P'\ H\ \land$
            $P = \textit{comp-et}\ J\ \land$
            $J = \textit{mset-set}\ T + \textit{mset-set}\ M\ \land$
            $M = \textit{comp-match}\ \{e \in E \mid e \subseteq W\}\ c\ \land$
```

```
        $W = \{v \in Vs\ T \mid \neg\ even'\ (degree\ T\ v)\} \wedge T = comp\text{-}mst\ c\ E\}$
$VAR\ \{0\}$
$DO\ v := hd\ P'$;
        $P' := tl\ P'$; $IF\ v \in H \wedge P' \neq []\ THEN\ SKIP\ \ ELSE\ H := v \cdot H\ FI$
$OD$
$\{is\text{-}hc\ E\ H\ \wedge$
 $(2 :: 'b) * cost\text{-}of\text{-}path\ (\lambda u\ v.\ c\ \{u,\ v\})\ H$
 $\leq (3 :: 'b) * cost\text{-}of\text{-}path\ (\lambda u\ v.\ c\ \{u,\ v\})\ OPT\}$
```

5 L-Reduction from Minimum Vertex Cover Problem to Metric TSP

This section describes the formalization of a L-Reduction from the Minimum Vertex Cover Problem, with maximum degree of 4 (VCP4), to the Metric TSP, which is presented in [6].

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.

Let A and B be optimization problems with cost functions c_A and c_B. The cost of the optimal solution for 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 $OPT(f\ x_A) \leq OPT(x_A)$, 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. $|(c_A\ x_A\ (g\ x_A\ y_B)) - OPT(x_A)| \leq |\ (c_A\ (f\ x_A)\ y_B) - OPT(f\ x_A)|$.

We L-reduce the VCP4 to the Metric TSP.

To describe an L-reduction, the definition of two functions f and g is required. For the function f, we are given an instance of the VCP4 and we need to construct an instance of the Metric TSP. An instance of the VCP4 consists of an arbitrary graph where the degree of any vertex is at most 4. To construct an instance of the Metric TSP we need to construct a complete graph with edge weights s.t. the edge weights satisfy the triangle-inequality.

The function f is described by the following construction. Let G be an instance of the VCP4, and let H:=f G. For each edge e of G we add a substructure H_e to H (see figure 1). The graph H is the complete graph over the vertices of the substructures H_e. The substructure H_e consists of 12

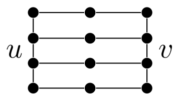

Figure 1: Substructure H_e for an edge $e{:=}\{u{,}v\}$.

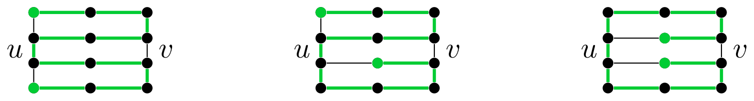

Figure 2: This figure shows the different types Hamiltonian paths for the substructure H_e for an edge $e{:=}\{u{,}v\}$. All three Hamiltonian paths correspond to the endpoint u.

vertices that are arranged in a 4-by-3 lattice structure. Each corner vertex corresponds to an endpoint of the edge e.

The substructure H_e for the edge e has the special property that there is no Hamiltonian path in H_e that starts and ends at corner-vertices of the substructure that correspond to different endpoints of the edge e. Therefore, the start- and end-vertex of a Hamiltonian path for the substructure H_e can only correspond to one endpoint of the edge e. This property is used to encode a vertex cover of G in a Hamiltonian cycle of H.

The substructure H_e admits 3 types of Hamiltonian paths (see figure 2).

Next, I describe the edge weights of the graph H. The weight of an edge between vertices of the same substructure H_e is simply the distance of the vertices within the substructure H_e. An edge that connects two corners of two different substructures H_e and H_f ($e \neq f$), where both corners are on the side of their substructure that corresponds to the same vertex have weight 4. All remaining edges have weight 5. The proof that the edge weights satisfy the traingle-inequality is omitted.

For the function g we need to construct a feasible solution for G given a feasible solution for H, which is an arbitrary Hamiltonian cycle T in H. A Hamiltonian cycle T in H may be composed of only Hamiltonian paths for the substructures H_e. In this case, for each edge e of G we look at the Hamiltonian path of H_e that is contained in T: the Hamiltonian path can only correspond to one endpoint of the edge e. We select this endpoint to be the covering vertex for the edge e. If a Hamiltonian cycle T in H does not contain a Hamiltonian path for every H_e, we construct a Hamiltonian cycle T' in 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 iteratively replacing parts of the current Hamiltonian cycle with a Hamiltonian path for a substructure 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. The locale *VC4-To-mTSP* additionally 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.

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

The function c uses a couple of auxiliary functions to check different cases for the two given vertices. The function *is-edge-in-He* checks if there is a substructure H_e where the two given vertices are connected by an edge. The function *are-vertices-in-He* checks if there is a substructure H_e that contains both given vertices. The function *min-dist-in-He* computes the length of the shortest path in a substructure H_e that connects the two given vertices.

```
$c\ G\ (e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2) =$
$($if $\textit{is-edge-in-He}\ G\ (\textit{uEdge}\ (e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2))$ then $1$
 else if $\textit{are-vertices-in-He}\ G\ (e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2)$
      then $\textit{the-enat}\ (\textit{min-dist-in-He}\ G\ (e_1,\ w_1,\ i_1)\ (e_2,\ w_2,\ i_2))$
      else if $\textit{rep1}\ e_1 \neq \textit{rep1}\ e_2\ \wedge$
              $w_1 = w_2 \wedge (i_1 = 1 \vee i_1 = 2) \wedge (i_2 = 1 \vee i_2 = 2)$
           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 has less total weight and the cycle contains a Hamiltonian path for every substructure H_e. The function *vc-of-tour* computes a vertex cover for G given a Hamiltonian cycle that contains a Hamiltonian path for every substructure H_e.

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

Please note, that this definition of the function g is a little different compared to the definition by [6]. In [6], the function g only inserts a Hamiltonian path for a substructure H_e if the current Hamiltonian cycle does not contain a Hamiltonian path for H_e. Thus, resulting Hamiltonian cycle may contain arbitrary Hamiltonian paths for the substructures H_e. There are Hamiltonian path for the substructures H_e that do not start or end at a corner.

This makes identifying the covering vertex for an edge more difficult. On the other hand, my definition of the function g makes sure that the resulting Hamiltonian cycle only consists of Hamiltonian paths for the substructures H_e that start and end at the corners of the substructures H_e. Therefore, identifying the covering vertex is simplified. Figure 2 depicts the different types of Hamiltonian paths in the substructure H_e.

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.

$g1.ugraph\text{-}adj\text{-}map\text{-}invar\ G \Longrightarrow g2.is\text{-}complete\text{-}Adj\ (f\ G)$

$g1.ugraph\text{-}adj\text{-}map\text{-}invar\ G \wedge 1 < |g1.uedges\ G| \wedge g2.is\text{-}hc\text{-}Adj\ (f\ G)\ T \Longrightarrow$
$g1.is\text{-}vc\text{-}Adj\ G\ (g\ G\ T)$

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

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

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.

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

This lemma is not proven in my formalization because I am unsure about the best way to prove this lemma. In the theory *tsp.FindHamiltonianPath* 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.

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 *tsp.MultiGraph*, 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] 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.

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