Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Theoretische Informatik

An Approximation Algorithm for the Asymmetric Travelling Salesman Problem

Bachelorarbeit

Autor:	Adrian Armstrong
Matrikelnummer:	10004856
Studiengang:	Informatik
Erstprüfer:	Prof. Dr. Heribert Vollmer
Zweitprüfer:	Dr. rer. nat. Arne Meier
Betreuer:	M. Sc. Anselm Haak
Abgabedatum:	30.01.2020

Contents

1. Introduction

The travelling salesman problem (TSP) is a well known combinatorial optimisation problem; its goal is to find a tour of minimum cost in an undirected graph given a weight function on the edges.

The corresponding decision problem ($\mathrm{TSP_D}$) of deciding whether a tour of cost at most some value K exists is NP-complete, therefore there is no known efficient algorithm for solving it and finding one would imply P=NP. In order to see why TSP is NP-hard the problem of deciding whether a Hamiltonian cycle exists in a graph can be reduced ($\leqslant_m^p$) to the problem of deciding whether a Hamiltonian cycle of cost at most some value K exists which is equivalent to $\mathrm{TSP_D}$. Karp [Kar72] showed that the Hamiltonian cycle problem is NP-complete which implies that $\mathrm{TSP_D}$ is NP-hard, it is also in NP because a certificate in form of an edge set can be verified in polynomial time by checking whether the solution is a valid tour and whether the total cost of the tour is at most K.

Therefore TSP (the optimisation problem) is NP-hard, which means that every problem in NP is Turing-reducible to TSP. Furthermore it is NP-hard to approximate TSP given arbitrary edge weights but there are polynomial time approximation algorithms for the so-called metric TSP, in which edge weights satisfy the triangle inequality $w(u,v) + w(v,w) \geqslant w(u,w)$ for vertices u, v and w. One such example is the $\frac{3}{2}$-approximation algorithm discovered by Christofides [Chr76].

In contrast, no constant-factor approximation algorithm was known for the asymmetric version of the travelling salesman problem (ATSP) until recently. ATSP takes a directed graph as input allowing for asymmetric edge weights, it is a more general version of TSP.

Definition 1.1. *The input for ATSP is a pair (G, w), where G is a strongly connected directed graph (digraph) and w is a nonnegative weight function defined on the edges satisfying the triangle inequality. The objective is to find a closed walk of minimum weight that visits every vertex at least once.*

We could also assume the graph is complete by letting the weight of an edge from u to v be equal to weight of the shortest path between u and v in the original graph. The new weights also satisfy the triangle inequality and this lets us skip vertices that were already visited in a walk like in the Christofides algorithm. Therefore we would not need to visit a vertex more than once. In this version of the problem we do not require the graph to be complete and vertices can be visited multiple times.

The reduction of A to B in the context of optimisation problems, means that a good approximation algorithm $\mathcal{A}$ for B can be turned into a good approximation algorithm for A, which uses $\mathcal{A}$ as a subroutine. In [Sve15] approximating ATSP is reduced to finding a so-called α-light solution to Local-Connectivity ATSP which will be presented

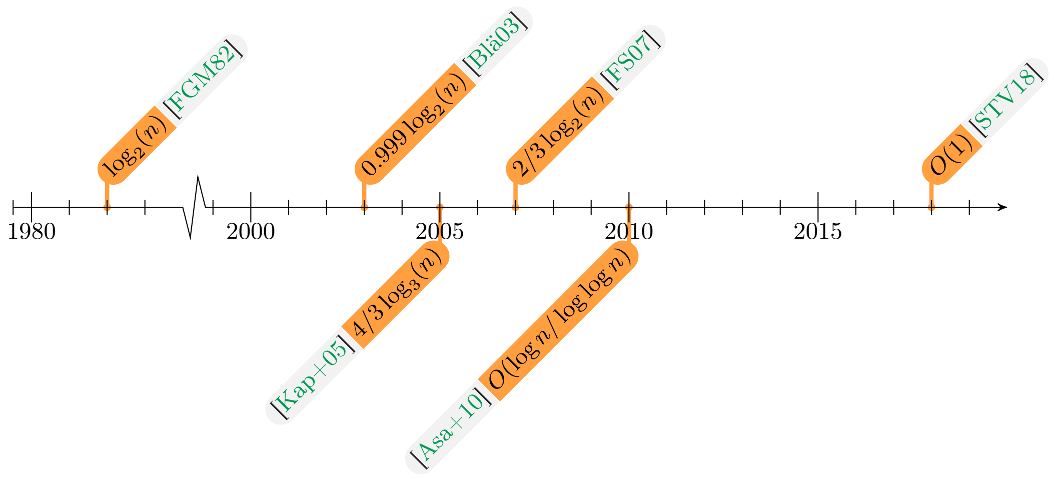

Figure 1.1.: a timeline of the best approximation guarantee for ATSP

in Chapter 6. Previously the best approximation guarantee for ATSP was in $O(\frac{\log n}{\log\log n})$ where n is the number of vertices [Asa+10]. Svensson, Tarnawski, and Végh [STV18] describe the first constant factor approximation algorithm for ATSP by reducing the problem of approximating ATSP to approximating ATSP on more structured instances called vertebrate pairs and then solving Local-Connectivity ATSP for vertebrate pairs resulting in an approximation algorithm for ATSP using the results in [Sve15].

In order to obtain more structured instances linear programming (discussed in Section 2.2) is used. Linear programming provides a mathematical model for optimisation problems in which the goal is to find an assignment of real numbers to some variables, subject to linear inequality constraints, such that a linear objective function is optimised. ATSP can be formulated as an integer linear program, in which the variables are constrained to be integers. These are NP-hard to solve in general, but if the integrality constraints of the variables are relaxed the problem can be solved in polynomial time, and the solution to the relaxed problem is a lower bound on the cost of an optimal tour. The relaxed linear program is called the *Held-Karp relaxation* (Section 3.1) and its optimal value is called the *Held-Karp lower bound*. Therefore an algorithm that finds a solution to ATSP with cost at most a constant α times the Held-Karp lower bound is also a α-approximation algorithm for ATSP.

The goal of this thesis is to clearly describe and explain the algorithm presented by Svensson, Tarnawski, and Végh [STV18]. Chapter 2 covers the necessary preliminaries on linear programming, while the following chapters present a series of reductions to easier problems. In Chapter 3 the problem is reduced to approximating ATSP on laminarly-weighted instances by using the Held-Karp relaxation. In Chapter 4 the problem is further reduced to approximating ATSP on irreducible instances via an algorithm that contracts certain vertex sets and then recursively solves the smaller instance. Chapter 5 provides a high level overview of the reduction to vertebrate pairs. Vertebrate pairs are instances together with a subtour B which is used to solve Local-Connectivity ATSP. In

Chapter 6 we present a more detailed treatment of the reduction from ATSP to Local-Connectivity ATSP. Finally the algorithm for Local-Connectivity ATSP is presented in Chapter 7, where we first focus on the special node-weighted case. The reader is assumed to have the basic knowledge of complexity theory taught by Meier [Mei18].

2. Preliminaries

In this chapter we introduce the notation that will be used throughout the thesis and present the basics of linear programming and network flow problems.

2.1. Notation

In this section we introduce the notation that will be used throughout the thesis. The notation is directly quoted for the most part and identical to that used by Svensson, Tarnawski, and Végh [STV18].

The *support* of a function $f\colon X \to \mathbb{R}_+$ is the subset $\{x \in X \mid f(x) > 0\}$. For a subset $Y \subseteq X$, we also use $f(Y) = \sum_{x \in Y} f(x)$. For multisets A, B we let their intersection $A \cap B$ be the multiset containing the elements that are contained in both A and B with multiplicity equal to the highest multiplicity in either set, e.g. $\{1, 1, 2\} \cap \{1, 2, 3\} = \{1, 1, 2\}$.

A *walk* is a sequence of edges where each edge is incident to the next. A walk that starts and ends in the same vertex is a *subtour*, and it is a *tour* if it visits every vertex at least once.

A directed graph $G = (V, E)$ is called *Eulerian* if the number of incoming edges is equal to the number of outgoing edges for every vertex. For a vertex set $U \subsetneq V$, we let $G[U]$ denote the subgraph induced by U. That is, $G[U]$ is the subgraph of G whose vertex set is U and whose edge set consists of all edges in the original edge set with both endpoints in U. We also let G/U denote the graph obtained by contracting the vertex set U, i.e., by replacing all the vertices in U by a single new vertex u and redirecting every edge with one endpoint in U to the new vertex u. This may create parallel edges in G/U. We keep all parallel copies; thus, every edge in G/U will have a unique preimage in G.

For vertex sets $S, T \subseteq V$ we let $\delta(S, T) = \{(u, v) \in E \mid u \in S \backslash T, v \in T \backslash S\}$. For a set $S \subseteq V$ we let $\delta^+(S) = \delta(S, V \backslash S)$ denote the set of outgoing edges, and we let $\delta^-(S) = \delta(V \backslash S, S)$ denote the set of incoming edges. Further, let $\delta(S) = \delta^-(S) \cup \delta^+(S)$. For a vertex $v \in V$ we let $\delta^+(v) = \delta^+(\{v\})$ and $\delta^-(v) = \delta^-(\{v\})$. An edge (multi)set F is called *Eulerian* if $|\delta^+(v) \cap F| = |\delta^-(v) \cap F|$ for all $v \in V$. A subtour is equivalent to an Eulerian multiset of edges that forms a single component.

For a set $S \subsetneq V$ we let S_{in} and S_{out} be those vertices of S that have an incoming edge from outside of S and those that have an outgoing edge to outside of S, respectively. That is,

$$S_{in} = \{v \in S \mid \delta^-(S) \cap \delta^-(v) \neq \emptyset\}, \text{ and } S_{out} = \{v \in S \mid \delta^+(S) \cap \delta^+(v) \neq \emptyset\}.$$

For $x = (x_1, \dots, x_n), y = (y_1, \dots, y_n) \in \mathbb{R}^n$ we say $x \leqslant y$ if and only if $x_i \leqslant y_i$ for $i = 1, \dots, n$.

2.2. Linear Programming

The following section is based on the textbook *Linear programming* by Chvátal [Chv83]. Linear programming (LP) is the problem of maximising or minimising the value of a linear function of n variables subject to m linear constraints. A linear program of the form

$$\begin{aligned} \text{maximise} \ & \sum_{j=1}^{n} c_n x_n, & \\ \text{subject to} \ & \sum_{j=1}^{n} a_{ij} x_j \leqslant b_i & (i = 1, 2, \dots, m), \\ & x_j \geqslant 0 & (j = 1, 2, \dots, n), \end{aligned}$$

is said to be in *standard form* and can also be written as

$$\begin{aligned} \text{max. } & c^T x, \\ \text{s.t. } & Ax \leqslant b, \\ & x \geqslant 0, \end{aligned}$$

where $c \in \mathbb{R}^n, b \in \mathbb{R}^m$ and $A = (a_{ij}) \in \mathbb{R}^{m \times n}$. The linear function $c^T x$ is called the *objective function.* The goal of this LP problem is to find a vector $x \in \mathbb{R}^n$ maximising the objective function and satisfying the constraints.

In general some of the constraints might be linear equations or inequalities of the form $ax \geqslant b$. Furthermore the goal might be to minimise the objective function instead of maximising it, which can be seen in the LP formulation of ATSP (Section 3.1). However, every such LP can be transformed into an equivalent problem in standard form using the following equivalences:

- min $c^T x \Leftrightarrow$ max $(-c)^T x$
- $ax \geqslant b \Leftrightarrow -ax \leqslant -b$
- $ax = b \Leftrightarrow ax \leqslant b \wedge -ax \leqslant -b$
- $x \geqslant 0$ missing $\Leftrightarrow$ add variables $x_{pos}, x_{neg} \geqslant 0$ and replace x with $x_{pos} - x_{neg}$

therefore we will only consider problems in standard form in this section.

A point x satisfying all of the constraints is called a *feasible solution* and the set $P = \{x \in \mathbb{R}^n \mid Ax \leqslant b,\ x \geqslant 0\}$ of all feasible solutions is called the *feasible region.* We call a linear program *feasible* if it has a feasible solution. For a linear inequality $a_1 x_1 + a_2 x_2 + \dots + a_n x_n \leqslant b$ the set of points $x \in \mathbb{R}^n$ satisfying the inequality is called a *half-space*. Furthermore the intersection of finitely many half-spaces is called a *convex polyhedron*, which may be unbounded. If the feasible region is unbounded then the value of the objective function on the feasible region may also be unbounded.

These terms stem from the following geometric interpretation. Geometrically the feasible region P of a linear program can be interpreted as a convex polyhedron, because it is the intersection of the half-spaces corresponding to the constraints (see Figure 2.1). Such a set is always convex which means that for any two points in P the line segment connecting the two points is also contained in P. Optimal solutions correspond to ver-

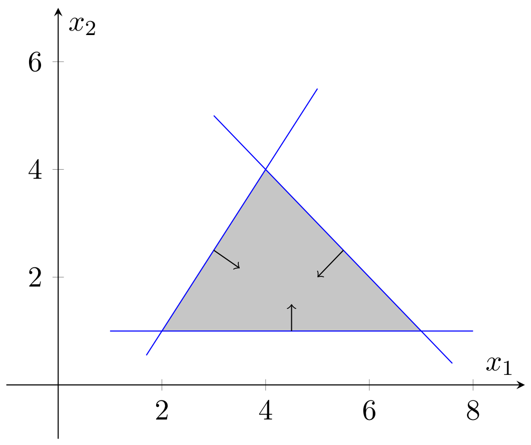

Figure 2.1.: geometric interpretation of a linear program $(n = 2, m = 3)$

tices of the polyhedron, or points on an edge (without the endpoints) in which case all the points along that edge (including the endpoints) are also optimal solutions. This means that if an optimal solution exists then there must be a vertex of the feasible region that is an optimal solution. One way of solving LP problems is the simplex method which checks the vertices by walking along the edges of the feasible region until an optimal solution is found, this is guaranteed to find an optimal solution if one exists but may take exponential time in the worst case, and the number of iterations increases proportionally to the number of constraints.

In theory linear programs can be solved efficiently using the ellipsoid method, which is discussed in Section 2.2.3. Even though it runs in polynomial time, the method is not well suited for practice due to numerical instability and slow convergence [BGT81]. In practice the simplex method is more effective even though it runs in exponential time in the worst case. Interestingly though, the ellipsoid method can be used to solve linear programs with exponentially many constraints in polynomial time.

2.2.1. Duality

For a linear program whose goal is to maximise the objective function any feasible solution x gives a lower bound $c^T x$ on the optimal value. Now consider the problem of finding an upper bound for the optimal value of the linear program

$$\begin{aligned}
\text{max. } & c_1x_1 + \cdots + c_nx_n, & \\
\text{s.t. } & a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \leqslant b_i & (i = 1, 2, \ldots, m), \\
& x_j \geqslant 0 & (j = 1, 2, \ldots, n).
\end{aligned}$$

Let $y_1, y_2, \ldots, y_m$ be variables. Multiplying the i'th constraint by y_i gives

$$\begin{gathered}
y_1(a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n) \leqslant y_1b_1 \\
\vdots \\
y_n(a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n) \leqslant y_nb_n,
\end{gathered}$$

and by adding the inequalities we get

$$\underbrace{(y_1a_{11} + y_2a_{21} + \cdots + y_na_{n1})}_{\text{constrain to be } \geqslant c_1} x_1 + \cdots + \underbrace{(y_1a_{1n} + y_2a_{2n} + \cdots + y_na_{nn})}_{\text{constrain to be } \geqslant c_n} x_n \leqslant \sum_{i=1}^{m} y_ib_y.$$

In order to find an upper bound the coefficients of the x_i on the left-hand side should be greater than or equal to their coefficients c_i the objective function. If the variables satisfy the above conditions and x is feasible, then clearly $\sum_{i=1}^{m} y_ib_y$ is an upper bound on c^Tx which leads to the *dual* problem of finding the tightest upper bound

$$\begin{aligned}
\text{minimise } & \sum_{j=1}^{m} b_ny_n, & \\
\text{subject to } & \sum_{j=1}^{n} a_{ji}y_j \leqslant b_i & (i = 1, 2, \ldots, n), \\
& y_j \geqslant 0 & (j = 1, 2, \ldots, m),
\end{aligned}$$

which can also be written as

$$\begin{aligned}
\text{min. } & b^Ty, \\
\text{s.t. } & A^Ty \geqslant c, \\
& y \geqslant 0.
\end{aligned}$$

the original problem is called the *primal* problem. If x and y are feasible primal and dual solutions, then we have

$$c^Tx \leqslant (A^Ty)^Tx = y^TAx \leqslant y^Tb, \tag{2.1}$$

where the first inequality holds because $c \leqslant A^Ty$ and the second holds because $Ax \leqslant b$.

Lemma 2.1. *Consider the linear program and its dual problem as defined above. If x and y are feasible primal and dual solutions such that $c^Tx = b^Ty$ then x and y are optimal solutions.*

Proof. Assume x and y are feasible solutions to the primal and dual problem respectively. The value of any solution to the dual is an upper bound on the optimal value of the primal therefore if $c^T x = b^T y$ and y is a feasible dual solution then $c^T x$ is an upper bound on the optimal value of the primal. But x was assumed to be a feasible solution therefore it must be optimal. □

Theorem 2.1 (duality)**.** *Consider the linear program and its dual problem as defined above. If the primal has an optimal solution $x^* \in \mathbb{R}^n$, then the dual has an optimal solution $y^* \in \mathbb{R}^m$ such that*

$$c^T x^* = b^T y^*.$$

Assume x^* is an optimal solution to the primal then if we can find a feasible solution y^* such that $c^T x^* = b^T y^*$ it is an optimal one by Lemma 2.1. The full proof of this theorem is not given here. The simplex method, which we will not discuss guarantees such a solution y^*.

Theorem 2.2 (complementary slackness)**.** *Let x^* and y^* be feasible solutions to the primal and dual problems respectively. Then x^* and y^* are optimal solutions if and only if the following conditions hold*

(i) $\sum_{i=1}^{m} a_{ij} y_i^* = c_j$ *or* $x_j^* = 0$ *(or both) for every* $1 \leqslant j \leqslant n$

(ii) $\sum_{j=1}^{n} a_{ij} x_j^* = b_i$ *or* $y_i^* = 0$ *(or both) for every* $1 \leqslant i \leqslant m$*.*

This means that the j'th constraint in the dual is tight or the corresponding primal variable is zero for every $1 \leqslant j \leqslant n$, and the i'th constraint in the primal is tight or the corresponding dual variable is zero $1 \leqslant i \leqslant m$.

Proof. Let x^* and y^* be feasible solutions. Then

$$\begin{aligned} x^* \text{ and } y^* \text{ are optimal } &\overset{2.1}{\iff} c^T x^* = b^T y* \\ &\overset{(2.1)}{\iff} c^T x^* = (A^T y^*)^T x^* = y^{*T} A x^* = y^* b^T \\ &\iff x_j^* = 0 \text{ or } c_j = \sum a_{ij} y_i^* \text{ and } y_i^* = 0 \text{ or } b_i = \sum a_{ij} x_j^* \end{aligned}$$

□

2.2.2. Integer Linear Programming

Many optimisation problems can be formulated as integer linear programs. Integer linear programming (ILP) is a variation of linear programming, where the variables and the entries in A, b and c are constrained to be integers. That is, a problem of the form

$$\begin{aligned} \text{max. } & c^T x, \\ \text{s.t. } & Ax \leqslant b, \\ & x \geqslant 0, \\ & x \in \mathbb{Z}^n. \end{aligned}$$

where $c \in \mathbb{Z}^n, b \in \mathbb{Z}^m$ and $A \in \mathbb{Z}^{m \times n}$.

The problem of 1-0 integer linear programming is a special case of ILP where the variables are constrained to be either one or zero. The problem of deciding whether an 1-0 ILP problem is feasible is one of Karp's 21 NP-complete problems [Kar72], where its NP-hardness is shown via reduction from SAT. Therefore, solving the optimisation version of such problems, and consequently ILP problems in general, is NP-hard. Certain ILP problems may be solvable in polynomial time, e.g. finding an integral circulation in a flow network (Section 2.3).

The LP problem where the integrality constraints on the variables are removed is called the *linear programming relaxation* of an ILP problem. Clearly the feasible region P_I of the ILP is contained within the feasible region P of the LP, because it consists of all those points in P whose entries are integers. Therefore the value of the optimal solution to the LP relaxation is an upper bound on the value of the optimal solution of the original problem.

While it is not known whether there is a polynomial time algorithm for solving ILP, the LP relaxation of such problems can be solved in polynomial time. One strategy of approximating ILP is to solve the LP relaxation and then round the solution to an integral one but this strategy depends on the "quality" of the LP relaxation, of which the *integrality gap* is an indication.

Definition 2.1 (integrality gap)**.** *Consider an optimisation problem with a set of instances I and a goal $t \in \{\min, \max\}$, that can be expressed using ILP. For an instance $x \in I$ of the problem let $\mathcal{OPT}_{\text{ILP}}(x)$ be the value of the optimal solution and $\mathcal{OPT}_{\text{LP}}(x)$ be the value of the optimal solution to the LP relaxation. Then*

$$\text{IG} = \begin{cases} \sup_{x \in I} \frac{\mathcal{OPT}_{\text{LP}}(x)}{\mathcal{OPT}_{\text{ILP}}(x)}, & \textit{for } t = \max \\ \sup_{x \in I} \frac{\mathcal{OPT}_{\text{ILP}}(x)}{\mathcal{OPT}_{\text{LP}}(x)}, & \textit{for } t = \min \end{cases}$$

is called the integrality gap. The integrality gap is at least one.

A high integrality gap means that the LP relaxation is in a sense, too optimistic and therefore it could be difficult to round a solution for the relaxed problem to a good integral solution. A low integrality gap (close to one) indicates a good LP relaxation.

In Section 3.1 the LP relaxation of the ILP formulation of ATSP called the Held-Karp relaxation is introduced. The best lower bound on the integrality gap for ATSP is 2 [CGK06] and previously the best upper bound for the integrality gap for ATSP was in $O(\text{poly} \log \log n)$ [AG14]. In practice it seems hard to find ATSP instances that have

$\frac{\mathcal{OPT}_{\mathrm{LP}}}{\mathcal{OPT}_{\mathrm{ILP}}}$ greater than 2. An α-approximation algorithm for ATSP with respect to its LP relaxation (3.1) implies an upper bound of α on the integrality gap for ATSP because such an algorithm guarantees an integral solution of weight at most $\alpha\mathcal{OPT}_{\mathrm{LP}} \geqslant \mathcal{OPT}_{\mathrm{ILP}}$ for every instance, and therefore $\frac{\mathcal{OPT}_{\mathrm{ILP}}}{\mathcal{OPT}_{\mathrm{LP}}} \leqslant \alpha$. The contraposition states that if IG $> \alpha$ then there exists no α-approximation algorithm with respect to the LP relaxation.

2.2.3. Ellipsoid Method

The ellipsoid method is an algorithm for deciding whether a linear program is feasible and can also be used to solve linear programs in polynomial time [Kha79]. This section explains the intuition behind this method and how it can be used to solve LP. A detailed description of the algorithm can be found in [KV12, Chapter 4]. Consider the linear program max. $c^T x$, s.t. $Ax \leqslant b, x \geqslant 0$ from the beginning. The ellipsoid method decides whether the feasible region $P = \{x \in \mathbb{R}^n \mid Ax \leqslant b,\ x \geqslant 0\}$ is empty and if not returns a vector $x \in P$. We will learn how this can be used to find an optimal solution.

The set of points $x \in \mathbb{R}^n$ satisfying a linear equation $a^T x = b$, where $a, b \in \mathbb{R}^n$ is called a *hyperplane*. The set of points $x \in \mathbb{R}^n$ satisfying an equation of the form $(x - v)^T A(x - v) \leqslant 1$, where $A \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^n$ is called an *ellipsoid*. Ellipsoids are the generalised n-dimensional version of ellipses.

In non-degenerate cases it is possible to bound the volume of P, so the algorithm starts with a ball E_0 containing P. In every iteration a smaller ellipsoid E_{k+1} is calculated, which is guaranteed to contain P. Eventually the center of the ellipsoid is a point in P, in which case the algorithm terminates, or the ellipsoid is too small, in which case P is empty and the problem is infeasible. Let the center of the current ellipsoid be $z \notin P$.

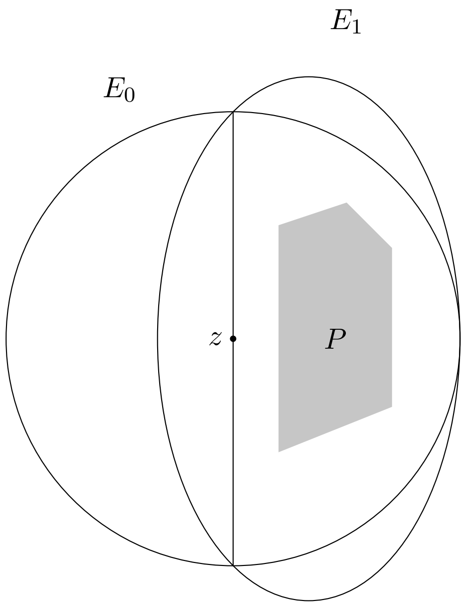

Figure 2.2.: one iteration of the ellipsoid method

Then there exists a hyperplane separating z from P, because P is convex. For at least one inequality we have $\sum_{j=1}^{n} a_{ij}z_j > b_i$ and therefore $\sum_{j=1}^{n} a_{ij}x_j \leqslant b_i \leqslant \sum_{j=1}^{n} a_{ij}z_j$ for all $x \in P$ because all points in P satisfy all of the inequalities. The hyperplane $H = \{x \mid \sum_{j=1}^{n} a_{ij}x_j = \sum_{j=1}^{n} a_{ij}z_j\}$, which goes through the point z and is parallel to the hyperplane corresponding to the broken inequality, separates z from P. The half-space $\{x \mid \sum_{j=1}^{n} a_{ij}x_j \leqslant \sum_{j=1}^{n} a_{ij}z_j\}$ corresponding to H contains P (see Figure 2.2). In order to check if a point is in P the method uses a separation oracle.

Definition 2.2. *Consider a linear program as defined at the beginning of the chapter. A separation oracle for that linear program is an algorithm which takes a vector $x \in \mathbb{R}^n$ as an input, and outputs whether x is a feasible solution. If not it returns a violated constraint*

$$\sum_{j=1}^{n} a_{ij}x_j > b_i \text{ for some } 1 \leqslant i \leqslant m.$$

In each iteration the ellipsoid E_{k+1} is constructed to contain the intersection of E_k with the half-space corresponding to a separating hyperplane. The number of iterations is in $O(poly(n, L))$ where L is the entry in A and b with the highest absolute value.

Surprisingly solving LP can be reduced to deciding feasibility. The ellipsoid method together with a separation oracle that runs in polynomial time can be used to solve LP problems in polynomial time. In order to see this define a new linear program with $n+m$ variables by combining the linear program with its dual:

$$\begin{aligned} \text{max. } & 0, \\ \text{s.t. } & Ax \leqslant b, \\ & A^T y \geqslant c, \\ & c^T x = b^T y, \\ & x \geqslant 0, \\ & y \geqslant 0. \end{aligned}$$

If this linear program is feasible, then any feasible solution (x^*, y^*) corresponds to feasible solutions to the primal and dual. Furthermore, because the condition $c^T x^* = b^T y^*$ holds, both x^* and y^* are optimal primal and dual solutions respectively due to Lemma 2.1. Therefore deciding feasibility and returning a feasible solution if the problem is feasible can be used to solve LP.

The problem with this method, is that the runtime of the ellipsoid method depends on the number of variables, and if the number of constraints m is exponential in the number of variables n then the runtime is exponential. It is possible to bound the absolute value of the optimal solution by some value $h \in \mathbb{R}$ based on L and n. Let $l = -h$. Another way of solving linear programs using the ellipsoid method is using binary search on the interval $[l, h]$ which contains the optimal value. First add the constraint $c^T x \geqslant \frac{l+h}{2}$ to the original linear program and use the ellipsoid method to check for feasibility. If the resulting linear program is feasible then the optimal value lies in the interval $[\frac{l+h}{2}, h]$, if not it lies in the interval $[l, \frac{l+h}{2}]$. The search terminates once the interval becomes too small to contain two distinct vertices with different objective values which happens after polynomially many iterations. Using this method the ellipsoid method can be used to solve LP problems with exponentially many constraints in polynomial time as long as there is polynomial time separation oracle for the problem.

2.3. Flow Problems

This section gives a brief overview of flow problems and is based on [Sch03].

Definition 2.3. *Let $G = (V, E)$ be a digraph, $s, t \in V$ and $c \colon V \to \mathbb{R}_+$ a capacity function. A function $f \colon V \to \mathbb{R}$ is called an $s - t$ flow subject to c if*

(i) $f(\delta^+(v)) = f(\delta^-(v))$ *for each* $v \in V \setminus \{s, t\}$

(ii) $0 \leqslant f(e) \leqslant c(e)$ *for each* $e \in E$.

The value of an $s - t$ flow f is

$$\mathrm{val}(f) = f(\delta^+(s)) - f(\delta^-(s)).$$

Definition 2.4. *Let $G = (V, E)$ be a digraph, $s, t \in V$ and $c \colon V \to \mathbb{R}_+$. A set $U \subsetneq V$ with $s \in U$ and $t \in V \setminus U$ is called an $s - t$ cut. The capacity of an $s - t$ cut is $c(\delta^+(U))$. If f is a flow subject to c then*

$$\mathrm{val}(f) \leqslant c(\delta^+(U)).$$

Definition 2.5. *Let $G = (V, E)$ be a digraph, $s, t \in V, c \colon V \to \mathbb{R}_+$ and f an $s - t$ flow subject to c. Define the edge set $E_f = \{e \in E \mid f(e) < c(e)\} \cup \{(v, u) \mid (u, v) \in E, f((u, v)) > 0\}$. $G_f = (V, E_f)$ is called the residual graph of f.*

Theorem 2.3 (max-flow min-cut)**.** *Let $G = (V, E)$ be a digraph, $s, t \in V$ and $c \colon V \to \mathbb{R}_+$. Then the maximum value of an $s - t$ flow subject to c is equal to the minimum capacity of an $s - t$ cut.*

Ford and Fulkerson provide an algorithm for finding a maximum flow [FF62]. A maximum flow in a flow network can be found in polynomial time using a variation of Ford and Fulkerson's algorithm [EK72]. If f is a maximum $s - t$ flow, then a minimum capacity $s - t$ cut is defined by the set of vertices reachable from s in the residual graph of f. Finding minimum $s - t$ cuts will be used to construct a separation oracle for the Held-Karp relaxation.

In a flow problem the amount of incoming flow through a vertex v can be constrained by splitting it into two vertices v_-, v_+ that are connected by an edge. Then all edges (u, v) are replaced by (u, v_-) and all edges (v, u') are replaced by (v_+, u'). Now a capacity can be enfored on the edge (v_-, v_+) which acts as a bottleneck limiting the flow on the edges in $\delta(v)$.

Circulation problems are a variant of flow problems, where there is no source or sink.

Definition 2.6. *Let $G = (V, E)$ be a digraph. A function $f\colon E \to \mathbb{R}$ is called a circulation if*

$$f(\delta^+(v)) = f(\delta^-(v)) \text{ for each } v \in V$$

Theorem 2.4 (Cycle decomposition)**.** *Let $G = (V, E)$ be a digraph and $f\colon E \to \mathbb{R}$ be a nonnegative circulation. Then f can be written as a nonnegative linear combination of at most $|E|$ directed cycles.*

This means that the circulation f is equal to the sum of certain circulations on a set of cycles in the graph. These cycles do not have to be disjoint. Flows can be written as a linear combination of $s - t$ paths and cycles.

Theorem 2.5 (Hoffman's circulation theorem)**.** *Let $G = (V, E)$ be a digraph and let $d, c\colon E \to \mathbb{R}$ with $d \leqslant c$. Then there exists a circulation f satisfying $d \leqslant f \leqslant c$ if and only if*

$$d(\delta(U)) \leqslant c(\delta(U)) \text{ for each } U \subseteq V.$$

If moreover d and c are integer, f can be taken integer.

Edmonds and Karp showed that a minimum cost circulation can be found in polynomial time [EK72].

Corollary 2.5.1. *Let $G = (V, E)$ be a digraph and let $f\colon E \to \mathbb{R}$ be a circulation. Then there exists an integer circulation f' with*

$$\lfloor f(e) \rfloor \leqslant f'(e) \leqslant \lceil f(e) \rceil \text{ for each } e \in E$$

Proof. Take $d := \lfloor f \rfloor$ and $c := \lceil f \rceil$ in Theorem 2.5. □

Integer circulations are used to solve Local-Connectivity ATSP in Chapter 7.

3. Reduction to Laminarly-Weighted Instances

In this chapter solving ATSP on general instances is reduced to solving ATSP on laminarly-weighted instances (defined in Section 3.2) by using linear programming. This allows us to focus on solving ATSP on these instances, which makes the problem easier because they carry additional information about the original instance.

In the following, let $(G = (V, E), w)$ be an ATSP instance with n vertices and m edges. Assuming the instance has a solution in the form of an edge multiset F, it can be written as a vector $x^* \in \mathbb{N}_0^m$. The vector has an entry x_e^* for every edge in E, which is to be interpreted as the number of times that edge is present in F. As mentioned in Chapter 2, ATSP can be modelled as an integer linear program that has a variable x_e for each edge. The following section introduces the LP relaxation of that ILP problem, in which the variables can take on positive real values. Even though it seems like this is of not much use, because an edge cannot be "taken" 0.5 times in a tour as the proverbial salesman cannot split himself in half, the duality of the linear program (Section 2.2.1) can be used to gain additional information about the instance. We will also discuss how to solve the LP relaxation which is called the Held-Karp relaxation in order to obtain a more structured instance.

3.1. Held-Karp Relaxation

The Held-Karp relaxation is the following linear program, which was first studied by Held and Karp [HK70; HK71]. Its optimal value is called the Held-Karp lower bound.

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} w(e)x(e) & & \qquad (\mathbf{LP}) \\ \text{subject to} \quad & x(\delta^+(v)) = x(\delta^-(v)) & & \text{for } v \in V, \\ & x(\delta(S)) \geqslant 2 & & \text{for } \emptyset \neq S \subsetneq V, \\ & x \geqslant 0. & & \end{aligned}$$

In this formulation x is also treated as a function on the edges, so $x(F) = \sum_{e \in F} x_e$ for some edge set F. The first set of constraints state, that the solution should be Eulerian. These constraints also imply $x(\delta^+(S)) = x(\delta^-(S))$ for $S \subseteq V$. The second set of constraints are called the *subtour-elimination constraints* of which there are $2^n - 2$, they guarantee that the solution is connected and does not consist of multiple disjoint subtours. An integral solution to the above linear program is a solution to ATSP because such a solution corresponds to an edge multiset that is a minimum cost tour of the graph.

The Held-Karp relaxation has $2n + 2^n - 2$ constraints and therefore the method of combining the problem with its dual and then using the ellipsoid algorithm from Section 2.2.3 is not suitable for solving it. In order to solve the problem in polynomial time using the ellipsoid method we need a separation oracle, which decides whether a given vector $x \in \mathbb{R}^m$ is a feasible solution to **LP**. The degree constraints on the vertices and the positivity constraints of the variables can easily be checked in polynomial time. In order to check the subtour-elimination constraints the given vector x is used as a capacity function on the edges, that is $c(e) = x_e$, for every $e \in E$ which results in a flow network. If at least one of the constraints is violated for some vertex subset $S \subsetneq V$ then the $s - t$ cut given by that subset has a capacity of less than one for some $s \in S, t \in V \setminus S$. Therefore $O(n^2)$ minimum $s - t$ cuts are computed in order to check the constraints, which can be done in polynomial time, leading to the following algorithm.

Algorithm 1 Separation oracle for the Held-Karp relaxation

```
input: a vector $x \in \mathbb{R}^m$

 1: for all $v \in V$ do
 2:     check the degree constraint $x(\delta^+(v)) = x(\delta^-(v))$
 3: end for
 4: for all $e \in E$ do
 5:     check the positivity constraint $x_e \geqslant 0$
 6: end for
 7: let the capacity $c$ of each edge $e \in E$ be defined as $c(e) = x_e$
 8: for all $(s, t) \in V \times V$ with $s \neq t$ do
 9:     compute a minimum $s - t$ cut $U$ in $G$ with the capacity function $c$
10:     if the capacity of the $s - t$ cut is less than 1 then
11:         return the subtour-elimination constraint corresponding to the cut $U$
12:     end if
13: end for
14: return $x$
```

The binary search method in Section 2.2.3, together with the separation algorithm can be used to solve the Held-Karp relaxation in polynomial time.

The dual problem to the Held-Karp relaxation is

$$\begin{aligned} &\text{maximise} \sum_{\emptyset \neq S \subsetneq V} 2 \cdot y_S && && \textbf{(DUAL)} \\ &\text{subject to} \sum_{S \in \{R \subsetneq V \mid (u,v) \in \delta(R)\}} y_S + \alpha_u - \alpha_v \leqslant w(u, v) && \text{for } (u, v) \in E, \\ &\qquad y \geqslant 0. \end{aligned}$$

The dual problem is obtained by transforming the primal (**LP**) as discussed in Section 2.2.1 and it has a variable for every constraint in the primal and therefore exponentially many (in the number of vertices n). Here the α_v variables correspond to the degree constraints, and the y_S variables correspond to the subtour-elimination constraints.

A linear program with exponentially many variables cannot be solved in polynomial time with the ellipsoid method, but the solution to the primal can be used to formulate an equivalent linear program with less variables. When solving the primal, polynomially many calls to the separation oracle are made, which returns a violated constraint each time. Consider a modified version of **LP**, where all (subtour-elimination) constraints are dropped, except those corresponding to a broken constraint returned by the separation oracle when solving the primal. The new linear program has polynomially many constraints, because the ellipsoid method only does polynomially many queries to the separation oracle. Because the ellipsoid method is deterministic and its result depends on the broken constraints returned by the separation oracle, the ellipsoid algorithm returns the same result for the modified linear program.

Due to Theorem 2.1 the optimal value of the new linear program is the same as that of its dual which is equal to the optimum value of the original **DUAL** problem. Therefore it suffices to find an optimal solution to the dual of the new linear program. The new dual can be solved in polynomial time, because it has polynomially many variables and constraints and they can easily be checked in polynomial time.

3.2. Laminarly-weighted ATSP and Uncrossing

In the previous section we learnt how the Held-Karp relaxation and its dual can be solved in polynomial time. This section introduces laminarly-weighted instances, which use an optimal solution to **LP** and a laminar optimal solution to **DUAL**.

Definition 3.1. *Let V be a finite set. A family of subsets $\mathcal{L} \subseteq \mathcal{P}(V)$ is called laminar if for any two sets $S, T \in \mathcal{L}$ either $S \cap T = \emptyset, S \subseteq T$ or $T \subseteq S$.*

We say a solution (y, α) to **DUAL** is laminar, if the support $\mathcal{L} = \{S \subsetneq V \mid y_S > 0\}$ of y is laminar.

Lemma 3.1. *There exists a laminar optimal solution to **DUAL** and it can be computed in polynomial time.*

Proof. Let (y^*, α^*) be an optimal solution to **DUAL**. Define a new linear program by adding the constraint

$$\sum_{\emptyset \neq S \subsetneq V} 2 \cdot y_S = \sum_{\emptyset \neq S \subsetneq V} 2 \cdot y^*_S \tag{3.1}$$

to **DUAL** and defining a new objective function to be minimized as

$$\sum_{\emptyset \neq S \subsetneq V} |S| y_S.$$

The new linear program can be solved in polynomial time using the method discussed in the previous section and the optimal solution (y, α) is laminar.

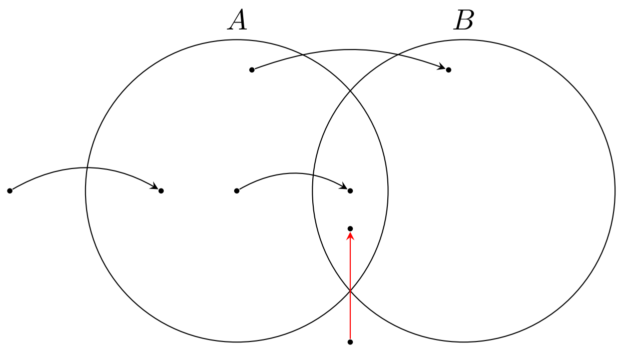

Figure 3.1.: Two crossing vertex sets.

Two sets $A, B \subsetneq V$ cross if they intersect in a nontrivial way, that is $A \cap B, A \setminus B$ and $B \setminus A$ are non-empty. If no sets in $\mathcal{L} = \{S \subsetneq V \mid y_S > 0\}$ cross, then it is laminar. Assume there are two sets $A, B \in \mathcal{L}$ that cross and suppose that, without loss of generality, $y_A \leqslant y_B$. Then obtain a new solution $(\tilde{y}, \alpha)$ to **DUAL** by modifying y in the following way

- $\tilde{y}_A = 0$ (removing it from $\mathcal{L}$)
- $\tilde{y}_B = y_B - y_A$
- $\tilde{y}_{B \setminus A} = y_{B \setminus A} + y_A$ (potentially adding it to $\mathcal{L}$)
- $\tilde{y}_{A \setminus B} = y_{A \setminus B} + y_A$ (potentially adding it to $\mathcal{L}$)
- $\tilde{y}_S = y_S$ for all the remaining subsets

The objective value of the new solution is the same as that of the old one, because y_A is both added and subtracted twice. Furthermore $\tilde{y}$ satisfies all of the constraints in **DUAL** because for all $(u, v) \in \delta(A \cap B, V \setminus (A \cup B))$ we have

$$\sum_{S \in \{R \subsetneq V \mid (u,v) \in \delta(R)\}} \tilde{y}_S = \sum_{S \in \{R \subsetneq V \mid (u,v) \in \delta(R)\}} y_S - 2y_A$$

because (u, v) crosses both A and B, but not $A \setminus B$ or $B \setminus A$ (see the red edge in Figure 3.1). For all other edges that cross A or B the sum stays the same, for example any edge in $\delta(A \setminus B, V \setminus (A \cup B))$ crosses A and $A \setminus B$ which means y_A was added and subtracted once. Therefore $(\tilde{y}, \alpha)$ is also an optimal solution to **DUAL**. However

$$\sum_{S \in \mathcal{L}} |S|(y_S - \tilde{y}_S) = (|A| + |B| - |A \setminus B| - |B \setminus A|)y_A > 0,$$

which is a contradiction to the fact that (y, α) is an optimal solution to the new linear program minimising $\sum_{\emptyset \neq S \subsetneq V} |S| y_S$. Therefore (y, α) is a laminar optimal dual solution. $\square$

Definition 3.2 (laminarly-weighted instances)**.** *A tuple $\mathcal{I} = (G, \mathcal{L}, x, y)$ is called a laminarly-weighted ATSP instance if G is a strongly connected digraph, $\mathcal{L}$ is a laminar family of vertex subsets, x is a feasible solution to* **LP***, and $y\colon \mathcal{L} \to \mathbb{R}_+$. We further require that $x_e > 0$ for every $e \in E$ and that every set $S \in \mathcal{L}$ be tight with respect to x, i.e. that $x(\delta^+(S)) = x(\delta^-(S)) = 1$. We define the induced weight function $w_\mathcal{I}\colon E \to \mathbb{R}_+$ as*

$$w_\mathcal{I}(e) = \sum_{S \in \{R \in \mathcal{L} | e \in \delta(R)\}} y_S \text{ for every } e \in E.$$

In the following it is shown, that an α-approximation algorithm for ATSP on laminarly-weighted instances can be turned into an α-approximation algorithm for general instances. Laminarly-weighted instances are easier to solve, because they contain additional structure and are a more general version of node-weighted instances in which the weight function is defined as $w(u, v) = f(u) + f(v)$ where $f\colon V \to \mathbb{R}_+$. Node weighted-instances correspond to laminarly-weighted instances, in which the laminar family $\mathcal{L}$ contains only singleton sets (containing only a single vertex) and Svensson [Sve15] provides an algorithm for Local-Connectivity ATSP on such instances (see Chapter 7).

Theorem 3.1. *Assume we have a polynomial-time algorithm that finds a solution of weight at most α times the Held-Karp lower bound for every laminarly-weighted ATSP instance. Then there is a polynomial-time algorithm for the general ATSP problem that finds a solution of weight at most α times the Held-Karp lower bound.*

Proof. Let x be an optimal solution to **LP** and (y, α) an optimal laminar solution to **DUAL** which can be computed using Lemma 3.1.

Define $E' = \{e \in E \mid x_e > 0\}$, $G' = (V, E')$ and $\mathcal{L} = \{S \subsetneq V \mid y_S > 0\}$. Then $\mathcal{I} = (G', \mathcal{L}, x, y)$ is a laminarly-weighted instance and due to complimentary slackness (Theorem 2.2) all of the constraints in the dual are tight for all edges e in E' and therefore the following holds for the induced weight function:

$$w_\mathcal{I}(u, v) = \sum_{S \in \{R \in \mathcal{L} | (u,v) \in \delta(R)\}} y_S = w(u, v) - \alpha_u + \alpha_v \text{ for } (u, v) \in E'.$$

We will show that a solution of weight at most α times the Held-karp Lower bound for $(G', w_\mathcal{I})$ is also an α-approximate solution in the original instance (G, w). Because x is a solution to **LP**, the corresponding edge multiset is Eulerian and therefore the α values cancel out in

$$\begin{aligned}
\sum_{e \in E'} w_\mathcal{I}(e) x(e) &= \sum_{(u,v) \in E'} \sum_{S \in \{R \in \mathcal{L} | (u,v) \in \delta(R)\}} y_S \cdot x(u, v) \\
&= \sum_{(u,v) \in E'} (w(u, v) - \alpha_u + \alpha_v) x(u, v) \\
&= \sum_{e \in E'} w(e) x(e).
\end{aligned}$$

It follows, that the optimal value to the Held-Karp relaxation for the instance $(G', w_\mathcal{I})$ is equal to the original Held-Karp lower bound. □

We define the LP value of a vertex set S to be

$$\text{value}_{\mathcal{I}}(S) = 2 \sum_{R \in \mathcal{L}, R \subsetneq S} y_R$$

The LP value of the entire vertex set $\text{value}_{\mathcal{I}}(V) = \text{value}(\mathcal{I})$ is equal to the Held-Karp lower bound.

3.3. Tight Sets

From now on Theorem 3.1 allows us to can focus on laminarly weighted instances of the form $\mathcal{I} = (G, \mathcal{L}, x, y)$. Due to complementary slackness (Theorem 2.2) all sets $\emptyset \neq S \subsetneq V$ with $y_S > 0$ have $x(\delta(S)) = 2$. Such sets are called *tight* and have certain nice properties.

The following lemmas given without proof provide some useful information about tight sets.

Lemma. *For a tight set $S \subsetneq V$ we have the following properties:*

(a) Every path from a vertex $u \in S_{in}$ to a vertex $v \in S_{out}$ (and thus every path traversing S) visits every strongly connected component of S.

(b) For every $u \in S_{in}$ and $v \in S$ there is a path from u to v inside S. The same holds for every $u \in S$ and $v \in S_{out}$.

Lemma. *Let $S \subsetneq V$ be a non-empty set such that $\mathcal{L} \cup \{S\}$ is a laminar family. Suppose $u, v \in S$ are two vertices such that there is a path from u to v inside S. Then we can in polynomial time find a path P from u to v inside S that crosses every set in $\mathcal{L}$ at most twice. Thus, the path satisfies $w(P) \leqslant \sum_{R \in \mathcal{L}: R \subsetneq S} 2 \cdot y_R = \text{value}(S)$.*

Remarkably, the new weight function $w_{\mathcal{I}}$, which is equal to the sum of the y values of all the sets crossed by an edge, is symmetric. Laminarly weighted instances contain alot of additional structure, but finding an algorithm for Local-Connectivity ATSP for such instances still seems too hard, therefore the problem is reduced even further in order to gain more structure.

4. Reduction to Irreducible Instances

In this chapter our problem is reduced to irreducible instances via an algorithm for laminarly-weighted instances that uses an algorithm for irreducible instances as a subroutine. This is done by contracting and inducing on tight vertex subsets in our instance, these operations are introduced in Section 4.1. After contracting a set, the resulting instance should ideally be easier to solve than the original instance and can therefore be solved recursively.

For a set $S \in \mathcal{L}$ and $u, v \in S$ we let

$$D_S(u,v) = \sum_{R \in \{A \in \mathcal{L} \mid u \in A, A \subsetneq S\}} y_R + d_S(u,v) + \sum_{R \in \{A \in \mathcal{L} \mid v \in A, A \subsetneq S\}} y_R,$$

where $d_S(u,v)$ is equal to the cost of a minimum weight path from u to v inside S, or $d_S(u,v) = \infty$ if no such path exists. For vetices $u \in S_{in}$ and $v \in S_{out}$, $2y_S + D_S(u,v)$ equals the cost of entering S via u, visiting all of the strongly connected components of S, and then exiting through v.

We let $\delta = 3/4$. This constant affects the approximation guarantee, and can be chosen differently in order to optimise the algorithm.

Definition 4.1. *We say that a set $S \in \mathcal{L}$ is reducible if*

$$\max_{u \in S_{in}, v \in S_{out}} D_S(u,v) < \delta \cdot \text{value}(S),$$

otherwise we say that S is irreducible. We also say that the instance $\mathcal{I}$ is irreducible if no set $S \in \mathcal{L}$ is reducible.

The intuition behind this is that irreducible instances are easier to solve because they resemble node-weighted instances in a sense. Laminarly-weighted instances where $\mathcal{L}$ only contains singletons are always irreducible, because contracting single vertices results in no decrease in LP value. Such instances are also node-weighted instances for which we already have a constant-factor approximation algorithm.

Reducible sets are sets, that result in a large decrease of the LP value of the instance when contracted (see Definition 4.2). Here we require the decrease to be at least $\frac{1}{4}$ value(S). The idea behind the reduction presented in Section 4.2 is that if there exists a reducible set S then it would be nice to contract it, solve the resulting smaller instance and then lift the solution back to a subtour of the original instance. Finally that subtour needs to be turned into a tour which can be done if we can find a tour of the instance obtained by inducing on the set S. In order for this to work, we need to assign an appropriate y-value to the vertex corresponding to the contracted set.

A smaller value of δ means that we require a larger reduction in LP value after contracting a set. Choosing a smaller value for δ would improve the approximation guarantee for laminarly-weighted instances (Algorithm 6) because the instances become alot easier after contracting a vertex set. However, it would also lead to a worse guarantee for irreducible instances (Algorithm 7) because more work has to be done in order to solve the instances that are considered irreducible. Furthemore, δ must be greater than one half.

4.1. Contracting and Inducing

Definition 4.2 (contracting a tight set)**.** *The instance $(G', \mathcal{L}', x', y')$ obtained from $\mathcal{I} = (G, \mathcal{L}, x, y)$ by contracting $S \in \mathcal{L}$, denoted by $\mathcal{I}/S$, is defined as follows:*

- *$G' = G/S$, let s denote the new vertex in G' corresponding to the set S*
- *$x'(e') = x(e)$ for each edge $e' \in E(G')$ where $e \in E(G)$ is the preimage of e' in G*
- *The laminar family $\mathcal{L}'$ contains all remaining sets of $\mathcal{L}$:*

$$\mathcal{L}' = \{R \setminus S \cup \{s\} \mid R \in \mathcal{L}, S \subseteq R\} \cup \{R \mid R \in \mathcal{L}, S \cap R = \emptyset\}$$

- *The vector y' equals y on all sets but $\{s\}$. For $\{s\}$ we define*

$$y'_s = y_S + \frac{1}{2} \max_{u \in S_{in}, v \in S_{out}} D_S(u, v)$$

For the value of the new instance obtained by contracting S we have

$$\text{value}(\mathcal{I}/S) = \text{value}(\mathcal{I}) - (\text{value}(S) - \max_{u \in S_{in}, v \in S_{out}} D_S(u, v)) \leqslant \text{value}(\mathcal{I}) \tag{4.1}$$

Definition 4.3. *For a tour T of $\mathcal{I}/S$ we define its lift to be the subtour of $\mathcal{I}$ obtained from T by replacing each consecutive pair $(u_{in}, s), (s, v_{out})$ of incoming and outgoing edges incident to s by their preimages (u_{in}, v_{in}) and (u_{out}, v_{out}) in G, together with a minimum-weight path from v_{in} to u_{out} inside S.*

Definition 4.4. *We say that $S \in \mathcal{L}$ is contractible with respect to an Eulerian multiset $F \subseteq E$ of edges if the lift of any tour of $\mathcal{I}/S$ plus the edge set F is a tour of $\mathcal{I}$.*

Lemma 4.1. *Let T be a tour of the instance $\mathcal{I}/S$. Then the lift F of T satisfies $w_{\mathcal{I}}(F) \leqslant w_{\mathcal{I}/S}(T)$.*

Definition 4.5 (inducing on a tight set)**.** *The instance $(G', \mathcal{L}', x', y')$ obtained from $\mathcal{I} = (G, \mathcal{L}, x, y)$ by inducing on $S \in \mathcal{L}$, denoted by $\mathcal{I}[S]$, is defined as follows:*

- *$G' = G/(V \setminus S)$, let $\overline{s}$ denote the new vertex in G' corresponding to the set $V \setminus S$*
- *$x'(e') = x(e)$ for each edge $e' \in E(G')$ where $e \in E(G)$ is the preimage of e' in G*

- *The laminar family $\mathcal{L}'$ contains $\{\bar{s}\}$ and all the sets that are strict subsets of S:*

$$\mathcal{L}' = \{R \in \mathcal{L} \mid R \subsetneq S\} \cup \{\{\bar{s}\}\}$$

- *The vector y' equals y on the sets common to $\mathcal{L}'$ and $\mathcal{L}$. For the new set $\{\bar{s}\}$ we define $y'_{\bar{s}} = \text{value}(S)$*

The new value is

$$\text{value}(\mathcal{I}[S]) = 2\,\text{value}(S). \tag{4.2}$$

Lemma 4.2. *Given a tour T of $\mathcal{I}[S]$, we can in polynomial time find an Eulerian multiset of edges $F \subseteq E$ such that S is contractible with respect to F and $w_{\mathcal{I}}(F) \leqslant w_{\mathcal{I}[S]}(T)$.*

4.2. Algorithm for Laminarly-Weighted Instances

This section contains the reduction to irreducible instances.

Theorem 4.1. *Let $\mathcal{A}_{irr}$ be a polynomial-time ρ-approximation algorithm for irreducible instances. Then there is a polynomial-time $\frac{2\rho}{1-\delta}$-approximation algorithm for general instances.*

For the proof, let $\mathcal{A}_{irr}$ be a polynomial-time ρ-approximation algorithm for irreducible instances. The reduction is given by the following algorithm, which works by selecting a reducible set containing no other reducible sets $R \subsetneq S$ in $\mathcal{L}$ and making it contractible via Lemma 4.2. Then S is contracted and the resulting instance $\mathcal{I}/S$ is solved recursively. The instance $\mathcal{I}[S]$ contains all subsets $R \subsetneq S \in \mathcal{L}$, which are all irreducible by the choice

Algorithm 2 $\mathcal{A}_{lam}$

```
   input: $\mathcal{I} = (G, \mathcal{L}, x, y)$
1: if $\mathcal{I}$ is irreducible then
2:     call $\mathcal{A}_{irr}$
3: else
4:     select a minimal reducible set $S \in \mathcal{L}$          ▷ $\mathcal{I}[S]$ is irreducible
5:     find a tour $T_S$ in $\mathcal{I}[S]$ using $\mathcal{A}_{irr}$
6:     find an edge set $F_S$ via Lemma 4.2 such that S is contractible w.r.t. $F_S$
7:     recursively call $\mathcal{A}_{lam}$ on $\mathcal{I}/S$
8:     return $F_S$ plus the lift of the resulting tour $T$ of $\mathcal{I}/S$
9: end if
```

of S, together with a set $\{\bar{s}\}$ which is irreducible because it is a singleton. Therefore $\mathcal{I}[S]$ is irreducible and the set S can be made contractible via Lemma 4.2, after using $\mathcal{A}_{irr}$ to find a tour of the instance.

After contracting S the resulting instance contains an additional set $\{s\} \in \mathcal{L}$, which is irreducible because it is a singleton. Furthermore for any superset $R \in \mathcal{L}$ of S the value of $\max_{u \in R_{in}, v \in R_{out}} D_R(u, v)$ can not decrease after contracting S. Thus the number of reducible sets in $\mathcal{L}$ decreases by at least one with each recursive call to $\mathcal{A}_{lam}$ eventually becoming zero. In this case an approximate tour can be found by simply calling $\mathcal{A}_{irr}$.

The algorithm terminates in polynomial time because at most $|\mathcal{L}|$ recursive calls to $\mathcal{A}_{lam}$ are made, in which both $\mathcal{A}_{irr}$ is called and Lemma 4.2 is invoked both requiring polynomial time.

Finally we prove that $\mathcal{A}_{lam}$ is a $\frac{2\rho}{1-\delta}$-approximation algorithm.

Base case: if $\mathcal{I}$ contains $n = 0$ reducible sets, then the algorithm calls $\mathcal{A}_{irr}$ which is a ρ-approximation algorithm.

Inductive step: Suppose the statement holds for instances with at most n reducible sets. Let $\mathcal{I}$ be an instance with $n + 1$ reducible sets and S be the minimal reducible set chosen by the algorithm. Then the algorithm returns the edge set $F_S \cup F$ where F is the lift of the tour T of $\mathcal{I}/S$ and F_S is an edge set making S contractible.

The edge set T_S was obtained by calling $\mathcal{A}_{irr}$ on $\mathcal{I}[S]$, therefore $w_{\mathcal{I}[S]}(T_S) \leqslant \rho\,\mathrm{value}(\mathcal{I}[S]) = 2\rho\,\mathrm{value}(S)$ and Lemma 4.2 guarantees that $w_{\mathcal{I}}(F_S) \leqslant w_{\mathcal{I}[S]}(T_S)$. The instance $\mathcal{I}/S$ has at most n reducible sets so by the induction hypothesis $w_{\mathcal{I}/S}(T) \leqslant \frac{2\rho}{1-\delta}\,\mathrm{value}(\mathcal{I}/S)$ and due to Lemma 4.1 we have $w_{\mathcal{I}}(F) \leqslant w_{\mathcal{I}/S}(T)$. Therefore

$$\begin{aligned} w_{\mathcal{I}}(F_S \cup F) &= w_{\mathcal{I}}(F_S) + w_{\mathcal{I}}(F) \\ &\leqslant w_{\mathcal{I}}(F_S) + w_{\mathcal{I}/S}(T) \\ &\leqslant 2\rho\,\mathrm{value}(S) + \frac{2\rho}{1-\delta}\,\mathrm{value}(\mathcal{I}/S) \\ &< 2\rho\,\mathrm{value}(S) + \frac{2\rho}{1-\delta}(\mathrm{value}(\mathcal{I}) - (1-\delta)\,\mathrm{value}(S)) \\ &= \frac{2\rho}{1-\delta}\,\mathrm{value}(\mathcal{I}) \end{aligned}$$

where the last inequality follows from (4.1) and the fact that S is a reducible set. This shows that the statement holds for instances with $n + 1$ irreducible sets completing the proof by induction. □

5. Reduction to Vertebrate Pairs

In this chapter the problem of approximating ATSP on irreducible instances is reduced to approximating ATSP on vertebrate pairs. We will not give any proofs instead the aim is to provide an overview of the reduction.

Definition 5.1. *We say that an Instance $\mathcal{I} = (G, \mathcal{L}, x, y)$ and a subtour B form a vertebrate pair if every $S \in \mathcal{L}$ with $|S| \geqslant 2$ is crossed by B, i.e., $\delta(S) \cap B \neq \emptyset$. The set B is referred to as the backbone of the instance.*

This problem resembles node-weighted instances even more. If the instance has no sets $S \in \mathcal{L}$ with $|S| \geqslant 2$ then the instance is node-weighted in which case Local-Connectivity ATSP is easy to solve. Otherwise the backbone crosses all such sets, which will be used in order to solve Local-Connectivity ATSP.

Theorem 5.1. *Let $\mathcal{A}_{ver}$ be a polynomial-time algorithm that, given a vertebrate pair $(\mathcal{I}', B)$, returns a tour of $\mathcal{I}'$ with weight at most $\kappa \operatorname{value}(\mathcal{I}') + \eta w_{\mathcal{I}'}(B)$(for some $\kappa, \eta \geqslant 0$). Then there is a polynomial-time ρ-approximation algorithm for irreducible instances, where $\rho = \frac{\kappa + \eta(\alpha_{nw}+3)}{1-2(1-\delta)}$.*

The input for the algorithm is an irreducible instance.

First a quasi-backbone is constructed which is a subtour that crosses a large fraction of the sets in $\mathcal{L}$ weighted by their LP value. The fraction of sets we require to be crossed depends on the choice of δ. The following lemma used to construct a quasi-backbone requires an approximation algorithm for node-weighted instances. Section 7.1 describes an algorithm for Local-Connectivity ATSP for such instances. The approximation guarantee of the algorithm for node weighted instances is $\alpha_{nw} = 18 + \epsilon$ which is obtained using the reduction to Local-Connectivity ATSP discussed in Chapter 6 (Theorem 7.1).

Lemma 5.1. *There is a polynomial-time algorithm that, given an irreducible instance $\mathcal{I} = (G, \mathcal{L}, x, y)$, constructs a quasi-backbone B such that $w(B) \leqslant (\alpha_{nw} + 3) \operatorname{value}(\mathcal{I})$ and $B \cap \delta(S) \neq \emptyset$ for every maximal non-singleton set $S \in \mathcal{L}$.*

The proof of this lemma will not be given. The quasi-backbone is constructed by contracting all maximal sets in $\mathcal{L}$ and then finding a tour B' in the resulting node-weighted instance. Then B' is lifted back to the original instance and modified such that it satisfies the desired properties.

After obtaining a quasi-backbone B all maximal non-singleton sets in $\mathcal{L}$ which are not crossed by B are contracted. Clearly B is a backbone in the resulting instance $\mathcal{I}'$ which can then be solved using the algorithm for vertebrate pairs. The output of the algorithm is the lift of the tour of $\mathcal{I}'$ together with a set of subtours making the sets in $\mathcal{I}$ that were contracted contractible. In order to make these vertex sets contractible the algorithm is recursively called on the instances induced by those sets which are also irreducible. The resulting tours of the induced instances are then used to make the vertex sets contractible.

The full algorithm can be found in Appendix A (Algorithm 7).

6. Reduction to Local-Connectivity ATSP

The following chapter is based on [Sve15] in which Svensson reduces the problem of approximating ATSP to the problem of finding a so-called α-light solution for Local-Connectivity ATSP which is defined in the following section. The reduction works for general ATSP instances (G, w) even though we only require an algorithm for vertebrate pairs in Theorem 5.1. Due to the generality of the reduction an algorithm for Local-Connectivity ATSP for vertebrate pairs implies an algorithm for ATSP for vertebrate pairs.

6.1. Local-Connectivity ATSP

Let $\mathcal{C}(E') = \{\tilde{G_1} = (\tilde{V_1}, \tilde{E_1}), \tilde{G_2} = (\tilde{V_2}, \tilde{E_2}), \dots \tilde{G_k} = (\tilde{V_k}, \tilde{E_k})\}$ denote the set of subgraphs corresponding to the k strongly connected components of the graph (V, E').

Let a lower bound function $\mathrm{lb}\colon V \to \mathbb{R}_+$ be fixed, so that $\mathrm{lb}(V) = \sum_{v \in V} \mathrm{lb}(v)$ is equal to the Held-Karp lower bound which is a lower bound on the cost of the optimal tour. Intuitively $\mathrm{lb}(v)$ expresses how much we would like to “pay” in order to visit a vertex v. In order for the reduction to work the function lb should also be efficiently computable. For a graph $\tilde{G}$ let $\mathrm{lb}(\tilde{G})$ denote the sum of the lb-values of the vertices in G.

Having defined a lower bound function we can now describe the quality of a subtour F visiting the vertices in $V' \subseteq V$ in terms of lb: $w(F)/\,\mathrm{lb}(V')$ should be small. This leads to the definition of Local-Connectivity ATSP:

Definition 6.1 (Local-Connectivity ATSP)**.** *The input for Local-Connectivity ATSP is a directed graph $G = (V, E)$, a weight function on the edges w and a partitioning of the vertices $V = V_1 \cup V_2 \cup \dots \cup V_k$ such that the graph induced by V_i is strongly connected for $i = 1, \dots, k$.*

The goal is to find an Eulerian multiset of edges F such that

$$|\delta^+(V_i) \cap F| \geqslant 1 \textit{ for } i = 1, \dots, k \textit{ and } \max_{(\tilde{V}, \tilde{E}) \in \mathcal{C}(F)} \frac{w(\tilde{E})}{\mathrm{lb}(\tilde{V})} \textit{ is minimised.}$$

The lower bound function can not have access to the partitioning.

A solution to Local-Connectivity ATSP is an Eulerian edge multiset that crosses all of the partitions; the solution does not have to be connected. We can think of this as a solution to ATSP where the subtour-elimination constraints that require connectivity are relaxed to only include the k constraints corresponding to the subsets in the partition.

The objective function says each connected component corresponding to a subtour in the solution should have a low cost. Consider, for example, the partition consisting of all the singleton sets $\{v\}, v \in V$ then a solution is a vertex cycle cover.

In the following section approximating ATSP is reduced to finding an α-light solution for Local-Connectivity ATSP. A solution F to Local-Connectivity ATSP is called α-light if it satisfies

$$\frac{w(\tilde{E})}{\mathrm{lb}(\tilde{V})} \leqslant \alpha \text{ for all subgraphs } (\tilde{V}, \tilde{E}) \in \mathcal{C}(F). \tag{6.1}$$

An algorithm for Local-Connectivity ATSP is called α-light, if it always finds a solution satisfying 6.1.

Finding an approximation algorithm for ATSP that finds a tour of weight at most α times the Held-Karp lower bound is at least as hard as finding an α-light algorithm for Local-Connectivity ATSP because such an algorithm guarantees a tour F with $w(F) \leqslant \alpha\,\mathrm{lb}(V)$ and thus $w(F)/\,\mathrm{lb}(V) \leqslant \alpha$. The tour visits every vertex and consists of only a single component therefore it is also an α-light solution to Local-Connectivity ATSP.

6.2. From Local to Global Connectivity

In this section the reduction is presented. Frieze, Galbiati, and Maffioli [FGM82] give a $\log_2 n$-approximation algorithm for ATSP by starting with a minimum cost cycle cover and then selecting a vertex in each component in that cycle cover. Then a minimum cost cycle cover is computed in the subgraph induced by those vertices, this is repeated until only one component is left. The number of connected components is at least halved each time and the weight of each cycle cover is at most the cost of an optimal tour resulting in a $\log_2 n$-approximate tour.

The idea behind this reduction uses a similar approach in that it starts with a cycle cover E^* and then iteratively adds cycles until $\mathcal{C}(E^*)$ contains only a single component. The connected components in $\mathcal{C}(E^*)$ partition the graph into k strongly connected components which is used as the input for Local-Connectivity ATSP. In each iteration cheap cycles together with a subset of the edges obtained by solving Local-Connectivity ATSP are added. The lower bound function can be chosen freely, as long it is fixed having no information about the partitioning and $\mathrm{lb}(V) = \mathrm{value}(V)$. The difficulty then lies in finding an α-light algorithm with the chosen lb function.

Theorem 6.1. *Let $\mathcal{A}$ be an algorithm for Local-Connectivity ATSP and consider an ATSP instance $G = (V, E, w)$. If $\mathcal{A}$ is α-light on G, there exists a tour of G with value at most $5\alpha\,\mathrm{lb}(V)$. Moreover, for any $\varepsilon > 0$, a tour of value at most $(9+\varepsilon)\alpha\,\mathrm{lb}(V)$ can be found in time polynomial in the number $n = |V|$ of vertices, in $1/\varepsilon$, and in the running time of $\mathcal{A}$.*

In the following let $\mathcal{A}$ be an α-light algorithm for Local-Connectivity ATSP. The first part of the theorem states, that a 5α-approximate tour exists and this section presents an algorithm for finding such a tour.

Definition 6.2. *We say that graphs $H_1 = (V_1, E_1), H_2 = (V_2, E_2), \ldots, H_k = (V_k, E_k)$ form an Eulerian partition of $G = (V, E)$ if the vertex sets $V_1, \ldots, V_k$ form a partition of V and each H_i is a connected Eulerian graph where E_i is a multisubset of E. An Eulerian partition is called α-light if*

$$w(E_i) \leqslant \alpha \operatorname{lb}(V_i) \quad \text{for } i = 1, \ldots, k$$

For the initialisation we start with a 2α-light Eulerian partition $H_1^* = (V_1^*, E_1^*), H_2^* = (V_2^*, E_2^*), \ldots, H_k^* = (V_k^*, E_k^*)$ with the following properties. The component H_1^* is chosen to be the connected component with the highest lb-value that has a tour of weight at most 2α times its lb-value and H_2^* is the largest such component which is disjoint from H_1^* and so on. This means that H_i^* is the connected component with the highest lb-value that has a tour of weight at most $2\alpha \operatorname{lb}(H_i^*)$ and is disjoint from the components $H_1^*, \ldots, H_{i-1}^*$. The partitions are ordered such that $\operatorname{lb}(V_1^*) \geqslant \operatorname{lb}(V_2^*) \geqslant \cdots \geqslant \operatorname{lb}(V_k^*)$. We do not have a polynomial time algorithm for finding such a partition because if a 5α-light tour exists, finding a 5α-light Eulerian partition satisfying the above properties would require finding a tour. However we can modify the algorithm later by starting with a cycle cover instead which results in a polynomial time algorithm with a slightly worse approximation ratio of $(9 + \epsilon)\alpha$.

Let the edge multiset $E^* = E_1^* \cup E_2^* \cup \cdots \cup E_k^*$ be the union of the edges of the partitions. Then $\mathcal{C}(E^*)$ contains the k components in the Eulerian partition. The total cost of the edges in E^* is at most $2\alpha \operatorname{lb}(V)$. In the following merge procedure certain partitions are connected by adding Eulerian sets of edges. This decreases the number of components in $\mathcal{C}(E^*)$ in every iteration and is repeated until E^* contains a tour. The cost of the tour can then be bounded by using the properties of the initial partitioning.

For a connected subgraph $\tilde{G}=(\tilde{E}, \tilde{V})$ of G, let $\operatorname{low}(\tilde{G}) = H^*_{\min\{i | V_i^* \cap \tilde{V} \neq \emptyset\}}$ which is the subgraph H_i^* in the Eulerian partition that intersects G maximising $\operatorname{lb}(V_i^*)$.

Algorithm 3 Merge procedure

```
   input: $E^*$
1: use $\mathcal{A}$ to find an Eulerian edge set $F$ crossing all of the partitions in $\mathcal{C}(E^*)$
2: remove edges in $F$ of connected components in $\mathcal{C}(F)$ that are completely contained
   in a component in $\mathcal{C}(E^*)$ (except trivial singleton components)
3: let $X = \emptyset$
4: select the component $\tilde{G}=(\tilde{E}, \tilde{V}) \in \mathcal{C}(F \cup X \cup E^*)$ that minimises $\operatorname{lb}(\operatorname{low}(\tilde{G}))$
5: if there exist a cycle $C=(V_C, E_C)$ in $G$ of weight $w(C) \leqslant \alpha \operatorname{lb}(\operatorname{low}(\tilde{G}))$ that connects
   $\tilde{G}$ to another component in $\mathcal{C}(E^* \cup F \cup X)$ then
6:     add $E_C$ to $X$ and goto 4
7: else
8:     $E^* \leftarrow E^* \cup (\tilde{E} \cap F) \cup (\tilde{E} \cap X)$
9: end if
```

We show that the merge procedure runs in polynomial time and reduces the number of connected components in $\mathcal{C}(E^*)$. The set of connected components can easily be computed in polynomial time using depth first search. In order to find a cycle connecting $\tilde{G}$ to another component (on line 5) consider the cycle consisting of an edge (u, v) together with a shortest path from v to u for all outgoing edges $(u, v) \in \delta^+(\tilde{V})$. Furthermore the algorithm starts with $k \leqslant n$ connected components, and therefore in each call to the algorithm at most k cycles can be found reducing the number of connected components in $\mathcal{C}(E^* \cup F \cup X)$.

The algorithm reduces the number of connected components in $\mathcal{C}(E^*)$ because in the final step it adds an Eulerian edge set $\tilde{E} \cap F$ and $\tilde{G}$ is a component in $\mathcal{C}(E^* \cup F \cup X)$. Furthermore, F is guaranteed to cross all of the components in $\mathcal{C}(E^*)$ so the edges in $\tilde{E} \cap F$ must connect at least two of the components.

Using the properties of the initial Eulerian partition $H_1^*, \dots, H_k^*$ we want to bound the weight of an Eulerian set of edges, which connect the partitions in order to form a tour. This can be done in the following two cases which will be used to bound the weight of the edges added in Algorithm 3.

The weight of a strongly connected Eulerian subgraph $\tilde{G} = (\tilde{V}, \tilde{E})$ with $w(\tilde{E}) \leqslant \alpha\, \mathrm{lb}(\tilde{V})$ can be bounded as follows:

$$\mathrm{lb}(\tilde{V}) \leqslant \mathrm{lb}(\mathrm{low}(\tilde{G})), \tag{6.2}$$

which implies $w(\tilde{E}) \leqslant \alpha\, \mathrm{lb}(\mathrm{low}(\tilde{G}))$.

Proof. Assume $\mathrm{lb}(\tilde{V}) > \mathrm{lb}(\mathrm{low}(\tilde{G}))$. We have $w(\tilde{E}) \leqslant \alpha\, \mathrm{lb}(\tilde{V}) \leqslant 2\alpha\, \mathrm{lb}(\tilde{V})$. If $\mathrm{low}(\tilde{G}) = H_i^*$ then $\tilde{G}$ is disjoint from $H_1^*, \dots, H_{i-1}^*$. This contradicts the fact that H_i^* was chosen to be the subgraph of highest lb-value containing a tour of weight at most $2\alpha\, \mathrm{lb}(H_i^*)$ that is disjoint from $H_1^*, \dots, H_{i-1}^*$. □

This means that any such Eulerian subgraph has weight at most α times the lb-value of the component in the partitioning with the highest lb-value that it intersects. Intuitively, this gives us a way to bound the weight of an α-light Eulerian set of edges that connects all of the partitions. If each of the partitions appears only once in the above inequality, then the total weight of the edges is at most $\alpha\, \mathrm{lb}(V)$. This would imply a tour the original instance of weight at most $(1 + 2)\alpha\, \mathrm{lb}(V)$ because the initial partitioning was 2α-light.

Furthermore, for disjoint Eulerian subgraphs $H_1 = (V_1, E_1), \dots, H_l = (V_l, E_l)$ with $w(E_j) \leqslant \alpha\, \mathrm{lb}(V_j)$ and $\mathrm{low}(H_j) = H_i^*$ for $j = 1, \dots, l$ we have

$$\sum_{j=1}^{l} \mathrm{lb}(V_j) \leqslant 2\, \mathrm{lb}(V_i^*) \tag{6.3}$$

This implies $\sum_{j=1}^{l} w(E_j) \leqslant 2\alpha\, \mathrm{lb}(V_i^*)$.

Proof. Assume $\sum_{j=1}^{l} \mathrm{lb}(V_j) > 2\, \mathrm{lb}(V_i^*)$. Consider the subgraph $H = (V', E')$ consisting of the union of H_i^* and $H_1, \dots, H_l$. Then H has $\mathrm{lb}(V') > \mathrm{lb}(V_i^*)$ furthermore, H is disjoint from $H_1^*, \dots, H_{i-1}^*$ because $\mathrm{low}(H_j) = H_i^*$ for $j = 1, \dots, l$. If $w(E') \leqslant 2\alpha\, \mathrm{lb}(V')$ this would contradict the fact that H_i^* was chosen to be subgraph maximising $\mathrm{lb}(H_i^*)$ that has a tour of weight at most $2\alpha\,\mathrm{lb}(H_i^*)$ and is disjoint from the subgraphs $H_1^*, \dots, H_{i-1}^*$. So we must have $2\alpha\,\mathrm{lb}(V') < w(E')$ and therefore

$$2\alpha \sum_{j=1}^{l} \mathrm{lb}(V_j) \leqslant 2\alpha\,\mathrm{lb}(V') < w(E')$$

and because $w(E') \leqslant \alpha \sum_{j=1}^{l} \mathrm{lb}(V_j) + 2\alpha\,\mathrm{lb}(V_i^*)$ we have

$$2\alpha \sum_{j=1}^{l} \mathrm{lb}(V_j) \leqslant \alpha \sum_{j=1}^{l} \mathrm{lb}(V_j) + 2\alpha\,\mathrm{lb}(V_i^*) \Leftrightarrow \sum_{j=1}^{l} \mathrm{lb}(V_j) \leqslant 2\,\mathrm{lb}(V_i^*)$$

which contradicts the assumption. □

This means that if we can connect all of the initial partitions by adding an Eulerian edge set F such that the partition with the lowest index that is crossed by any component in $\mathcal{C}(F)$ is H_i^*. Then the total weight of those edges is at most $2\alpha\,\mathrm{lb}(V_i^*)$. Similarly to the above case, this would imply a tour of the original instance of weight at most $(2+2)\alpha\,\mathrm{lb}(V)$.

Now we can use the inequalities (6.2) and (6.3) in order to show that the weight of the tour gained by repeatedly calling the merge procedure is at most $5\alpha\,\mathrm{lb}(V)$. Lets say $T \leqslant k$ calls to the merge procedure are required before finding a tour. Let $\tilde{G}_t = (\tilde{V}_t, \tilde{E}_t)$ be the current selected component and F_t, X_t be the selected edge sets after the else branch is taken in the t-th call to the merge procedure (on line 8) for $1 \leqslant t \leqslant T$. Further, let $\tilde{F}_t = F_t \cap \tilde{E}_t, \tilde{X}_t = X_t \cap \tilde{E}_t$ be the edges added to E^* each call.

Lemma 6.1. *We have $w(\cup_{t=1}^{T} \tilde{X}_t) \leqslant \alpha\,\mathrm{lb}(V)$.*

Proof. Each $\tilde{X}_t$ consists of a set of cycles in the component $\tilde{G}_t$ selected in the final iteration in the merge procedure. Let $C_1, C_2, \dots, C_c$ be the cycles in $\cup_{t=1}^{T} \tilde{X}_t$.

Lets say $\mathrm{low}(\tilde{G}_t) = H_i^*$ and C_j is the first cycle that connects $\tilde{G}_t$ to another component $G' = (V', E')$. Then $\mathrm{lb}(\mathrm{low}(G')) > \mathrm{lb}(V_i^*)$ because $\tilde{G}_t$ was selected to minimise $\mathrm{lb}(\mathrm{low}(\tilde{G}))$ (we can assume the inequality is strict without loss of generality). Furthermore, in the following steps G' and $\tilde{G}_t$ stay connected. Therefore any new component $\tilde{G}_{t'}$ that is selected later on cannot have $\mathrm{low}(\tilde{G}_{t'}) = H_i^*$, because low is by definition the partition intersecting our component maximising its lb-value and G' already intersects a component $(\mathrm{low}(G'))$ of higher lb-value.

The cycle was chosen to have $w(C_j) \leqslant \alpha\,\mathrm{lb}(V_i^*)$ (line 5) and this together with the fact that each V_i^* appears at most once by the reasoning above implies that $w(\cup_{j=1}^{c} C_j) = w(\cup_{t=1}^{T} \tilde{X}_t) \leqslant \alpha \sum_{i=1}^{k} \mathrm{lb}(V_i^*) = \alpha\,\mathrm{lb}(V)$. □

Lemma 6.2. *We have $w(\cup_{t=1}^{T} \tilde{F}_t) \leqslant 2\alpha\,\mathrm{lb}(V)$.*

Proof. Let $\mathcal{F}^t$ denote the set of the Eulerian subgraphs corresponding to the connected components in $\mathcal{C}(\tilde{F}_t)$ where we disregard the trivial components that only consist of a single vertex. Define $\mathcal{F}_i^t = \{H \in \mathcal{F}^t \mid \text{low}(H) = H_i^*\}$. Then $w(\tilde{F}_t) = w(\mathcal{F}^t) = \sum_{i=1}^k w(\mathcal{F}_i^t)$.

We now show that for each $i = 1, \dots, k$ the set $\mathcal{F}_i^t$ is non-empty for at most one $t = 1, \dots, T$.

Proof. Let $1 \leqslant i \leqslant k$ and $1 \leqslant t_0 < t_1 \leqslant T$. Assume that $\mathcal{F}_i^{t_0}, \mathcal{F}_i^{t_1} \neq \emptyset$.

The partition H_i^* is contained within $\tilde{G}_{t_0}$ because $\tilde{G}_{t_0} \in \mathcal{C}(F \cup X \cup E^*)$ and $\mathcal{F}_i^{t_0}$, which is contained in $\tilde{G}_{t_0}$, intersects H_i^*. Therefore $\text{lb}(\text{low}(\tilde{G}_{t_0})) \geqslant \text{lb}(V_i^*)$.

Let $H = (\hat{V}, \hat{E})$ be a component in $\mathcal{F}_i^{t_1}$. Then H cannot be contained within $\tilde{G}_{t_0}$ because otherwise it would have been removed from F in step 2 of the t_1-th call to the merge procedure. We also have $w(\hat{E}) \leqslant \alpha\, \text{lb}(\hat{V}) \leqslant \alpha\, \text{lb}(V_i^*)$ where the first inequality is due to the fact that F is α-light and the second is due to equation (6.2).

This means that H connects $\tilde{G}_{t_0}$ to another component in $\mathcal{C}(F \cup X \cup E^*)$ and $w(\hat{E}) \leqslant \alpha\, \text{lb}(V_i^*) \leqslant \alpha\, \text{lb}(\text{low}(\tilde{G}_{t_0}))$. And therefore there must exists a cycle C in H of weight at most $\alpha\, \text{lb}(\text{low}(\tilde{G}_{t_0}))$ that connects $\tilde{G}_{t_0}$ to another component. This is a contradiction because $\tilde{G}_{t_0}$ is the selected component after the else branch is taken in the t_0-th call to the merge procedure which means there is no such cycle, otherwise the if branch would have been taken. ■

Each $H = (V', E') \in \mathcal{F}_i^t$ is Eulerian and has $w(E') \leqslant \alpha\, \text{lb}(V')$ because F is an α-light solution to Local-Connectivity ATSP. Therefore $\text{lb}(\mathcal{F}_i^t) \leqslant 2\, \text{lb}(V_i^*)$ by fact (6.3) (here $\text{lb}(\mathcal{F}_i^t)$ denotes the sum of the lb values of all the components in $\mathcal{F}_i^t$). Using this we have

$$w(\cup_{t=1}^T \tilde{F}_t) = \sum_{t=1}^T \sum_{i=1}^k w(\mathcal{F}_i^t) \leqslant \alpha \sum_{t=1}^T \sum_{i=1}^k \text{lb}(\mathcal{F}_i^t) \leqslant 2\alpha \sum_{i=1}^k \text{lb}(V_i^*) = 2\alpha\, \text{lb}(V)$$

where the last inequality follows from the fact that $\mathcal{F}_i^t \neq \emptyset$ for at most one call to the merge procedure. □

Finally using Lemma 6.1 and 6.2 the weight of the tour is

$$w(\cup_{t=1}^T \tilde{F}_t) + w(\cup_{t=1}^T \tilde{X}_t) + \sum_{i=1}^k w(E_i^*) \leqslant (2 + 1 + 2)\alpha\, \text{lb}(V) = 5\alpha\, \text{lb}(V).$$

While the merge procedure runs in polynomial time, we do not have an algorithm for finding the 2α-light Eulerian partition in the initialisation. In order to get a polynomial time algorithm we start with the partitioning consisting of the sets $\{v\}, v \in V$, which is trivially 3α-light. Therefore E^* is initially empty and $\mathcal{C}(E^*)$ contains the subgraphs consisting of single vertices. As mentioned at the start solving Local-Connectivity ATSP for this partitioning results in a cycle cover. Then we run a modified merge procedure on the partitioning where the requirement for the cycles added to X is relaxed, until we have a tour. And in each call to the merge procedure we verify that the condition (6.4) holds. The following lemma which we will not prove guarantees that the resulting tour has weight at most $(9 + \epsilon)\alpha\, \text{lb}(V)$.

Lemma 6.3. *Assume that the algorithm is initialised with a 3α-light Eulerian partition $H_1^*, H_2^*, \ldots, H_k^*$ so that, in each repetition t of the modified merge procedure, we add a subset $\tilde{F}_t$ such that*

$$\mathrm{lb}(\mathcal{F}_i^t) < 3\ \mathrm{lb}(H_i^*) + \frac{\varepsilon\,\mathrm{lb}(V)}{n} \text{ for } i = 1, 2, \ldots, k. \tag{6.4}$$

Then the returned tour has weight at most $(9 + 2\varepsilon)\alpha\,\mathrm{lb}(V)$.

Here $\mathcal{F}_i^t$ is defined as in the proof of Lemma 6.2. If condition (6.4) is violated in any repetition of the modified merge procedure, then the entire algorithm is restarted with an Eulerian partition guaranteed by the following lemma, given without proof. This process is repeated until the condition is satisfied in each repetition of the merge procedure, resulting in a tour of the desired weight.

Lemma 6.4. *Suppose that repetition t of the (modified) merge procedure violates condition (6.4) when run starting from a 3α-light Eulerian partition $H_1^*, H_2^*, \ldots, H_k^*$. Then we can, in time polynomial in n, find a new 3α-light Eulerian partition $\hat{H}_1^*, \hat{H}_2^*, \ldots, \hat{H}_{\hat{k}}^*$ so that*

$$\sum_{j=1}^{\hat{k}} \mathrm{lb}(\hat{H}_j^*)^2 - \sum_{j=1}^{k} \mathrm{lb}(H_j^*)^2 \geqslant \frac{\varepsilon^2}{3n^2}\,\mathrm{lb}(V)^2 \tag{6.5}$$

Lets say we start with the partition $H_1^*, H_2^*, \ldots, H_k^*$ and denote the sum $\sum_{j=1}^{k} \mathrm{lb}(H_j^*)^2$ of the squared lb values by S_0. We will let S_l denote the sum of the squares of the lb values of the partition after invoking Lemma 6.4 for the l-th time.

If a repetition of the modified merge procedure violates the condition (6.4) after starting with that partition then Lemma 6.4 guarantees a second Eulerian partition with $S_1 \geqslant \frac{\varepsilon^2}{3n^2}\,\mathrm{lb}(V)^2 + S_0$. If a repetition of the merge procedure violates the condition again, then the lemma guarantees a third partition with

$$\frac{\varepsilon^2}{3n^2}\,\mathrm{lb}(V)^2 \leqslant S_2 - S_1 \leqslant S_2 - \left(\frac{\varepsilon^2}{3n^2}\,\mathrm{lb}(V)^2 + S_0\right) \Leftrightarrow \frac{2\varepsilon^2}{3n^2}\,\mathrm{lb}(V)^2 + S_1 \leqslant S_2$$

and so on. This means that the sum S_l increases by $\frac{\varepsilon^2}{3n^2}\,\mathrm{lb}(V)^2$ every time the algorithm is restarted, however for any Eulerian partition $H_1, H_2, \ldots, H_k$ we have $\sum_{i=1}^{k} \mathrm{lb}(H_i)^2 \leqslant (\sum_{i=1}^{k} \mathrm{lb}(H_i))^2 = \mathrm{lb}(V)^2$. Therefore we can find such a new Eulerian partition at most $3n^2/\varepsilon^2$ times, or we would have a partition with $S \geqslant \mathrm{lb}(V)^2 + S_0$ which is a contradition.

Furthermore the algorithm guarantees a new partition if the condition (6.4) is ever violated in a repetition of the merge procedure. Therefore the algorithm has to be restarted at most $3n^2/\varepsilon^2$ times before terminating without ever violating the condition.

The final algorithm for vertebrate pairs, together with the modified merge procedure can be found in Appendix A (Algorithm 8).

7. Algorithm for Local-Connectivity ATSP

In the previous chapters approximating ATSP was reduced to approximating ATSP on vertebrate pairs, which contain additional structure making Local-Connectivity ATSP easier to solve. The final problem is to find an α-light algorithm for Local-Connectivity ATSP on vertebrate pairs $(\mathcal{I}, B)$, which implies an approximation algorithm for ATSP on vertebrate pairs using Theorem 6.1.

Note that the reduction to Local-Connectivity ATSP in Chapter 6 works for general ATSP instances and does not take the additional structure provided by vertebrate pairs into account. In the algorithm for vertebrate pairs (Algorithm 8), however, $\mathcal{A}_{loc}$ is only called on vertebrate pairs so the backbone can be used in order to solve Local-Connectivity ATSP. Because the backbone already crosses all the non-singleton sets in $\mathcal{L}$ the problem becomes similar to the node-weighted case.

Definition 7.1 (node-weighted ATSP)**.** *The input for node-weighted ATSP is a strongly connected directed graph $G = (V, E)$ and a weight function $w \colon E \to \mathbb{R}_+$ such that there exists a weight function $f \colon V \to \mathbb{R}_+$ satisfying $w(u, v) = f(u)$. The goal is to find a tour of minimum weight.*

Equivalently we can require $w(u, v) = h(u) + h(v)$ by setting $h(u) = \frac{1}{2} f(u)$. The weight of any tour is the same because it has to be Eulerian.

7.1. Singleton instances

Laminarly-weighted instances $\mathcal{I}$ where $\mathcal{L}$ contains only sets with one element are node-weighted instances because any edge (u, v) crosses at most two sets in $\mathcal{L}$, namely $\{u\}$ and $\{v\}$ therefore $w_{\mathcal{I}}(u, v) = y_u + y_v$. Before looking at the algorithm for arbitrary vertebrate pairs, we prove the following theorem which provides a 2-light algorithm for these singleton instances.

Theorem 7.1. *There exists a polynomial time 2-light algorithm for Local-Connectivity ATSP on singleton instances.*

The input is a node-weighted instance $\mathcal{I} = (G, \mathcal{L}, x, y)$ together with a partitioning $V = V_1 \cup V_2 \cup \cdots \cup V_k$. For the lower bound function define $\mathrm{lb}(v) = 2y_v$. The output is a Eulerian edge set F satisfying

$$|\delta^+(V_i) \cap F| \geqslant 1 \text{ for } i = 1, \ldots, k \text{ and } \frac{w(\tilde{E})}{\mathrm{lb}(\tilde{V})} \leqslant \alpha \text{ for all subgraphs } (\tilde{V}, \tilde{E}) \in \mathcal{C}(F).$$

The vector x is a circulation because it satisfies the degree constraints in the linear program **LP**. In the following x will be modified and then rounded to an integral circulation satisfying the above conditions.

This algorithm adds a vertex A_i for all partitions V_i and then redirects exactly one unit of flow from x to go through A_i (see Figure 7.1). This is always possible because the subtour-elimination constraints in **LP** guarantee that $x(\delta^-(V_i)) \geqslant 1$. Furthermore $x(\delta^-(v)) = 1$ holds for all $v \in V$ with $y_v > 0$ due to complementary slackness.

Algorithm 4 redirecting flow

```
    input: circulation $x$, partitioning $V = V_1 \cup V_2 \cup \cdots \cup V_k$
    output: modified (multi)graph $G'$ and circulation $x'$
 1: for $i = 1, \dots, k$ do
 2:     add a vertex $A_i$
 3:     for all $(u, v) \in \delta^-(V_i)$ do                ▷ redirect one unit of flow to $A_i$
 4:         add an edge $(u, A_i)$
 5:         $x'(u, v) \leftarrow x(u, v)\left(1 - \frac{1}{x(\delta^-(V_i))}\right)$
 6:         $x'(u, A_i) \leftarrow x(u, v)\left(\frac{1}{x(\delta^-(V_i))}\right)$
 7:     end for
 8:     for all $(u, v) \in \delta^+(V_i)$ do                ▷ redirect one unit of flow from $A_i$
 9:         add an edge $(A_i, v)$
10:         $x'(u, v) \leftarrow x(u, v)\left(1 - \frac{1}{x(\delta^-(V_i))}\right)$
11:         $x'(A_i, v) \leftarrow x(u, v)\left(\frac{1}{x(\delta^-(V_i))}\right)$
12:     end for
13:     for all $(u, v)$ with $u, v \in V_i$ do                ▷ restore flow conservation
14:         $x'(u, v) \leftarrow x(u, v)\left(1 - \frac{1}{x(\delta^-(V_i))}\right)$
15:     end for
16: end for
```

The result x' is a circulation in G' because flow conservation holds for all vertices. Consider the i'th iteration of Algorithm 4. The flow on all of the edges having at least one endpoint in the set V_i is multiplied by a factor of $(1 - 1/x(\delta^-(V_i)))$. For all edges (u, v) in $\delta^-(V_i)$ the outgoing flow of u does not change because we add a flow of $x(u, v)(1/x(\delta^-(V_i)))$ on the edge (u, A_i). For all edges (u, v) in $\delta^+(V_i)$ the incoming flow of v does not change because we add a flow of $x(u, v)(1/x(\delta^-(V_i)))$ on the edge (A_i, v). Therefore the incoming and outgoing flow of a vertex $v \in V_i$ is $x(\delta^-(v))\,(1 - 1/x(\delta^-(V_i)))$. We now have $x'(\delta^-(A_i)) = 1$ for all $i = 1, \dots, k$ and $x'(\delta^-(v)) \leqslant 1$ still holds for all $v \in V$ with $y_v > 0$.

Recall that degree constraints can be enforced on a vertex v in a flow problem by splitting v into two vertices v_-, v_+ and then enforcing the degree constraints as capacities or lower bounds on the edge (v_-, v_+) (see Figure 7.1). This means we can enforce an upper bound of one on the incoming flow for each $v \in V$ with $y_v > 0$ as well as an upper and lower bound of one for the A_i. We can now round x' to an integral circulation z' in polynomial time using Corollary 2.5.1. The circulation z' satisfies the same degree bounds as x' because the degree constraints were already integral. The degree bounds on vertices with $y_v = 0$ do not matter, because these vertices can be visited for free as we can write the cost of an edge (u, v) as $w(u, v) = 2y_v$.

The circulation z' in G' is then mapped back to G by adding the flow on the edges entering and exiting A_i back onto the original edges. Note that G' may have parallel edges so that all edges (u, A_i) and (A_i, v) have a unique preimage in $\delta^-(V_i)$ and $\delta^+(V_i)$, respectively. Now the partition V_i has at least one unit of incoming flow from one of the edges going to the vertex A_i as well as one unit of outgoing flow from one of the edges coming from A_i.

The resulting flow z in G may not be a circulation because the in-degree and out-degree may not be equal for two vertices in each V_i. Let the incoming edge (u_{in}, v_{in}) and outgoing edge (u_{out}, v_{out}) be the edges that got one unit of flow added to them. If $v_{in} = u_{out}$ then flow conservation holds, if not, in order to fix this calculate a path P_i from v_{in} to u_{out} inside V_i. The output F is the multiset containing each edge e with multiplicity $z(e)$ together with the paths P_i.

Clearly all the partitions are crossed by F, because each of the vertices A_i had one unit of incoming flow before mapping back to the original graph. Furthermore each vertex v with $y_v > 0$ is visited at most twice because a degree constraint of one was enforced on v and the flow from the A_i together with the path P_i add at most one visit to each vertex in V_i. Therefore for any $(\tilde{V}, \tilde{E}) \in \mathcal{C}(F)$ we have

$$w_{\mathcal{I}}(\tilde{E}) = \sum_{e\in\tilde{E}} w_{\mathcal{I}}(e) = \sum_{v\in\tilde{V}} |\delta(v) \cap \tilde{E}| y_v = 2\sum_{v\in\tilde{V}} |\delta^-(v) \cap \tilde{E}| y_v \leqslant 4\sum_{v\in\tilde{V}} y_v = 2\,\mathrm{lb}(\tilde{V}).$$

Using Theorem 6.1 this implies an $(18+\epsilon)$-approximation algorithm for Local-Connectivity ATSP which is required for the reduction to vertebrate pairs in Theorem 5.1.

7.2. Non-Singleton Instances

Vertebrate pairs may contain non-singleton sets in their laminar family $\mathcal{L}$ and this makes them harder to solve, but they also contain a backbone B which crosses all non-singleton sets in $\mathcal{L}$, therefore the algorithm is designed to taking this additional structure into account.

The first step in solving Local-Connectivity ATSP is designing a lower bound function $\mathrm{lb}\colon V \to \mathbb{R}_+$ such that $\mathrm{lb}(V)$ equals the Held-Karp lower bound. Define the lower bound function

$$\mathrm{lb}(v) = \mathrm{value}(\mathcal{I}) \cdot \frac{\overline{\mathrm{lb}}(v)}{\overline{\mathrm{lb}}(V)}, \text{ where } \overline{\mathrm{lb}}(v) = \begin{cases} \frac{1}{|V(B)|}(\mathrm{value}(\mathcal{I}) + w(B)/4) & \text{if } v \in V(B), \\ 2y_v & \text{otherwise.} \end{cases}$$

The idea is to round the circulation x to an integral one like in the singleton case while additionally forcing at least one unit of flow to visit a vertex of the backbone whenever a

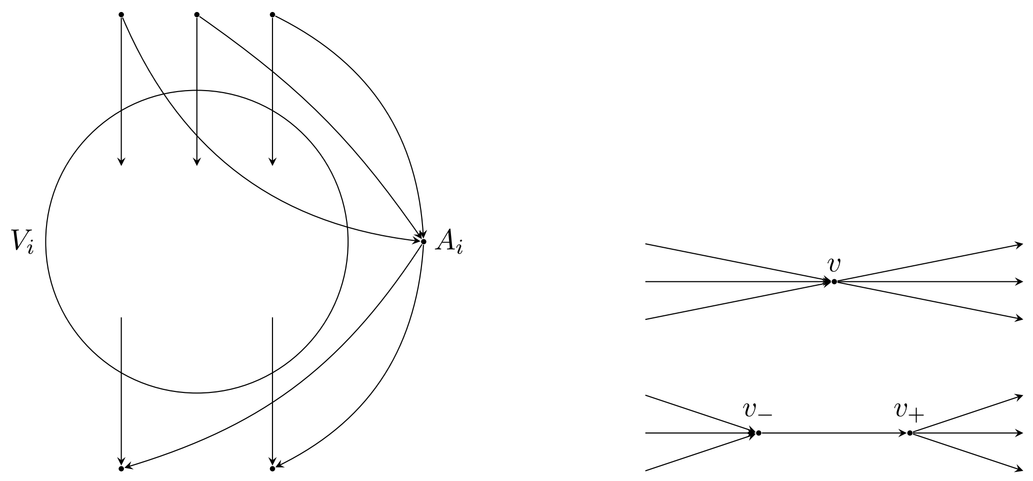

Figure 7.1.: On the left a fraction of the total flow is redirected to go through A_i. On the right a vertex is split into two, enforcing a capacity on the edge (v_-, v_+) is the same as enforcing a degree bound on v.

non-singleton set is crossed by the circulation. This results in one large component of the solution containing the backbone together with all of the edges crossing a non-singleton component, this means that the lower bound value of that component is at least $\mathrm{lb}(B)$. The analysis for the remaining components in the solution is similar to the singleton case.

Theorem 7.2. *There is a polynomial-time algorithm for Local-Connectivity ATSP that is $(8 + w(B)/\operatorname{value}(\mathcal{I}))$-light for vertebrate pairs $(\mathcal{I}, B)$.*

The input for Local-Connectivity ATSP is a partitioning $V = V_1 \cup V_2 \cup \cdots \cup V_k$. The output is an Eulerian edge multiset $F^* = B \cup P \cup F$.

Because the instance already has a backbone B which crosses all the sets $S \in \mathcal{L}$ with $|S| \geqslant 2$ it is added as part of the output F^*. Any partition V_i that is crossed by the backbone therefore already has $|\delta^+(V_i) \cap F^*| \geqslant 1$. If the backbone is completely contained inside a partition then we add a subtour P connecting the backbone to any other partition otherwise let $P = \emptyset$.

The set F crosses the remaining partitions that are not already crossed by the backbone. It is obtained using the following lemma, which is given without proof.

Lemma 7.1. *There is a polynomial-time algorithm that solves the following problem. Let $(\mathcal{I}, B)$ be a vertebrate pair, and let $U_1, \ldots, U_l \subseteq V \backslash V(B)$ be disjoint non-empty vertex sets such that the subgraphs $G[U_1], \ldots, G[U_l]$ are strongly connected and for every $S \in \mathcal{L}_{\geqslant 2}$ and $i = 1, \ldots, l$ we have either $U_i \cap S = \emptyset$ or $U_i \subseteq S$. Then the algorithm finds an Eulerian multiset $F \subseteq E$ of edges such that:*

(a) $w(F) \leqslant 3 \cdot \operatorname{value}(\mathcal{I})$,

(b) $|\delta^-_F(U)| \geqslant 1$ *for every* $i = 1, \ldots, l$,

(c) $|\delta_F^-(v)| \leqslant 4$ *for every* $v \in V$ *with* $x(\delta^-(v)) = 1$,

(d) any subtour in F *that crosses a tight set in* $\mathcal{L}_{\geqslant 2}$ *visits a vertex of the backbone.*

This allows us to construct the required low cost Eulerian set of edges by choosing the U_i as specific subsets of the partitions V_i that aren't crossed by the backbone. The full algorithm can be found in Appendix A (Algortihm 9). This implies a $(9+\epsilon)(8+w(B)/\operatorname{value}(\mathcal{I}))$-approximation algorithm for vertebrate pairs.

8. Conclusion

The final approximation guarantee of the algorithm presented here is 4177. In the latest version of the paper Svensson, Tarnawski, and Végh [STV17] improve the constant down to 506 by optimising the reductions. By introducing a specialised version of Local-Connectivity, the fact that $\mathcal{A}_{loc}$ (Algorithm 9) is always called on vertebrate pairs can be exploited. Instead of using the reduction from [Sve15] which provides a general reduction from ATSP to Local-Connectivity ATSP, they reduce solving ATSP for vertebrate pairs to this new problem, which takes the backbone into account.

A 506 approximation algorithm for ATSP with respect to the Held-Karp relaxation gives an upper bound of 505 on the integrality gap of ATSP because it guarantees a solution F of weight $w(F) \leqslant 506\,\mathrm{value}(\mathcal{I})$ for every instance $\mathcal{I}$ and therefore $\mathcal{OPT} \leqslant 506\,\mathrm{value}(\mathcal{I})$ which implies $\mathrm{IG} = \frac{\mathcal{OPT}}{\mathrm{value}(\mathcal{I})} \leqslant 506$ for every instance. This is improved down to 319 by constructing a non polynomial time approximation algorithm with a better approximation guarantee using the first part of Theorem 6.1.

In order to beat the $\log_2 n$-approximation algorithm of Frieze, Galbiati, and Maffioli [FGM82] (see Figure 1.1) the number of vertices n would have to be greater than 2^{506} therefore the algorithm is not suitable for practical use. However, achieving a constant factor is an important theoretical advancement. Improving on the methods used in [STV18] in order to get a better approximation guarantee is an open problem, but in order to get close to the conjectured integrality gap of 2, new methods are probably necessary.

A. Full Description of Algorithm

The following series of reductions was made

1. Algorithm for laminarly-weighted instances $\Rightarrow$ algorithm $\mathcal{A}$ for general instances
2. Algorithm for irreducible instances $\Rightarrow$ algorithm $\mathcal{A}_{lam}$ for laminarly-weighted instances
3. Algorithm for vertebrate pairs $\Rightarrow$ algorithm $\mathcal{A}_{irr}$ for irreducible instances
4. Algorithm for Local-Connectivity ATSP $\Rightarrow$ algorithm $\mathcal{A}_{ver}$ for vertebrate pairs

All of the algorithms used in the reduction steps are listed below. The final algorithm $\mathcal{A}_{loc}$ solves Local-Connectivity ATSP for vertebrate pairs.

Algorithm 5 $\mathcal{A}$

```
    input: $G = (V, E), w\colon V \times V \to \mathbb{R}_+$
 1: find optimal solutions $x$ and $(\alpha, y)$ to LP and DUAL
 2: if $y$ is not laminar then
 3:     $y \leftarrow$ a laminar optimal solution obtained via uncrossing $y$ (Section 3.2)
 4: end if
 5: let
 6:     $E' = \{e \in E(G) \mid x(e) > 0\}$
 7:     $G' = (V, E')$
 8:     $\mathcal{L} = \{S \subsetneq V \mid y_S > 0\}$
 9:     $w'(u, v) = w(u, v) - \alpha_u + \alpha_v$
10: call $\mathcal{A}_{lam}$ on $\mathcal{I} = (G', \mathcal{L}, x, y)$          ▷ $w_{\mathcal{I}} = w'$
```

Algorithm 6 $\mathcal{A}_{lam}$

```
    input: $\mathcal{I} = (G, \mathcal{L}, x, y)$
 1: if $\mathcal{I}$ is irreducible then
 2:     call $\mathcal{A}_{irr}$
 3: else
 4:     select a minimal reducible set $S \in \mathcal{L}$          ▷ $\mathcal{I}[S]$ is irreducible
 5:     find a tour $T_S$ in $\mathcal{I}[S]$ using $\mathcal{A}_{irr}$
 6:     find an edge set $F_S$ via Lemma 4.2 such that S is contractible w.r.t. $F_S$
 7:     recursively call $\mathcal{A}_{lam}$ on $\mathcal{I}/S$
 8:     return $F_S$ plus the lift of the resulting tour $T$ of $\mathcal{I}/S$
 9: end if
```

Algorithm 7 $\mathcal{A}_{irr}$

```
    input: irreducible instance $\mathcal{I} = (G, \mathcal{L}, x, y)$
 1: construct a quasi-backbone $B$ using Lemma 5.1
 2: let $\mathcal{L}^*_{max}$ be the set of all maximal non-singleton sets in $\mathcal{L}$ that $B$ does not cross
 3: for all $S \in \mathcal{L}^*_{max}$ do
 4:     recursively call $\mathcal{A}_{irr}$ on $\mathcal{I}[S]$ to find a tour $T_S$ of $\mathcal{I}[S]$
 5:     find an edge set $F_S$ via Lemma 4.2 making S contractible
 6: end for
 7: let $\mathcal{I}'$ be the instance obtained by contracting all $S \in \mathcal{L}^*_{max}$
 8: Assert: B is now a backbone of $\mathcal{I}'$
 9: call $\mathcal{A}_{ver}$ on $(\mathcal{I}', B)$ resulting in a tour $T'$ of $\mathcal{I}'$
10: return the lift of $T'$ together with $\bigcup_{S \in \mathcal{L}^*_{max}} F_S$
```

Algorithm 8 $\mathcal{A}_{ver}$

```
    input: vertebrate pair $(\mathcal{I}, B)$
 1: start with each vertex as a subgraph (trivially $3\alpha$-light and Eulerian)
 2: $E^* \leftarrow \emptyset$ (the edges of the initial Eulerian partitions)
 3: while condition (6.4) holds and there is more than one partition do
 4:     run merge on the partitioning
 5: end while
 6: if (6.4) is violated then
 7:     restart with a new partitioning via Lemma 6.4
 8: end if
 9: return $E^*$
10: function MERGE(partitioning)
11:     use $\mathcal{A}_{loc}$ to find an Eulerian edge set $F$ crossing all of the partitions
12:     remove edges in $F$ of connected components in $\mathcal{C}(F)$ that are completely contained
    in a component in $\mathcal{C}(E^*)$ except trivial singletons
13:     let $X = \emptyset$
14:     select the component $\tilde{G}=(\tilde{E},\tilde{V})\in\mathcal{C}(F \cup X \cup E^*)$ that minimises $\mathrm{lb}(\mathrm{low}(\tilde{G}))$
15:     if there exist a cycle $C=(V_C, E_C)$ in $G$ of weight $w(C) \leqslant \alpha(3\,\mathrm{lb}(\mathrm{low}(\tilde{G})) +$
    $\varepsilon\,\mathrm{lb}(V)/n)$ that connects $\tilde{G}$ to another component in $\mathcal{C}(E^* \cup F \cup X)$ then
16:         add $E_C$ to $X$ and goto 14
17:     else
18:         $E^* \leftarrow E^* \cup (\tilde{E} \cap F) \cup (\tilde{E} \cap X)$
19:     end if
20: end function
```

Algorithm 9 $\mathcal{A}_{loc}$ for Local-Connectivity ATSP

```
    input: partitioning $V = V_1 \cup V_2 \cup \cdots \cup V_k$, backbone $B$
    output: $B \cup P \cup F$ where $P$ and $F$ are obtained as follows
 1: if there is no partition class that contains all the vertices of $B$ then
 2:     let $P = \emptyset$
 3: else
 4:     let $V_i$ be the partition class with $V(B) \subseteq V_i$
 5:     select arbitrary vertices $u \in V(B)$ and $v \in V \backslash V_i$
 6:     let $P$ be the edges of a minimium-weight path from $u$ to $v$ plus the edges of a
        minimium-weight path from $v$ to $u$.
 7: end if
 8: index the partition classes so that $V_1, \ldots, V_l$ are those sets that are disjoint from the
    vertices of the backbone (we have $0 \leqslant l < k$)
 9: for $i = 1, \ldots, l$ do
10:     let $V_i'$ be the intersection of $V_i$ with a minimal set $S \in \mathcal{L}_{\geqslant 2} \cup \{V\}$ with $S \cap V_i \neq \emptyset$
11:     consider a decomposition of $V_i'$ into strongly connected components
12:     let $U_i \subseteq V_i'$ be the vertex set of a source component in this decomposition
13: end for
14: let $F$ be the edge set guaranteed by Lemma 7.1
```

Bibliography

[AG14] Nima Anari and Shayan Oveis Gharan. *Effective-Resistance-Reducing Flows, Spectrally Thin Trees, and Asymmetric TSP*. 2014. arXiv: `1411.4613 [cs.DS]` (on page 9).

[Asa+10] Arash Asadpour et al. "An O(log n/log log n)-approximation Algorithm for the Asymmetric Traveling Salesman Problem". In: *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, Jan. 2010 (on page 2).

[BGT81] Robert G. Bland, Donald Goldfarb, and Michael J. Todd. "The Ellipsoid Method: A Survey". In: *Operations Research* 29 (1981), pp. 1039–1091 (on page 6).

[Blä03] Markus Bläser. "A New Approximation Algorithm for the Asymmetric TSP with Triangle Inequality". In: *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '03. Baltimore, Maryland: Society for Industrial and Applied Mathematics, 2003, pp. 638–645. ISBN: 0-89871-538-5 (on page 2).

[CGK06] Moses Charikar, Michel X. Goemans, and Howard Karloff. "On the Integrality Ratio for the Asymmetric Traveling Salesman Problem". In: *Mathematics of Operations Research* 31.2 (May 2006), pp. 245–252 (on page 9).

[Chr76] 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 (on page 1).

[Chv83] Vašek Chvátal. *Linear programming*. New York: W.H. Freeman, 1983 (on page 5).

[EK72] Jack Edmonds and Richard M. Karp. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems". In: *J. ACM* 19.2 (Apr. 1972), pp. 248–264 (on pages 12, 13).

[FF62] D. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton, NJ, USA: Princeton University Press, 1962 (on page 12).

[FGM82] A. M. Frieze, G. Galbiati, and F. Maffioli. "On the worst-case performance of some algorithms for the asymmetric traveling salesman problem". In: *Networks* 12.1 (1982), pp. 23–39 (on pages 2, 27, 38).

[FS07] Uriel Feige and Mohit Singh. “Improved Approximation Ratios for Traveling Salesperson Tours and Paths in Directed Graphs”. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques.* Springer Berlin Heidelberg, 2007, pp. 104–118 (on page 2).

[HK70] Michael Held and Richard M. Karp. “The Traveling-Salesman Problem and Minimum Spanning Trees”. In: *Operations Research* 18.6 (Dec. 1970), pp. 1138–1162 (on page 14).

[HK71] Michael Held and Richard M. Karp. “The traveling-salesman problem and minimum spanning trees: Part II”. In: *Mathematical Programming* 1.1 (Dec. 1971), pp. 6–25 (on page 14).

[Kap+05] Haim Kaplan et al. “Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs”. In: *Journal of the ACM* 52.4 (July 2005), pp. 602–626 (on page 2).

[Kar72] Richard M. Karp. “Reducibility among Combinatorial Problems”. In: *Complexity of Computer Computations.* Springer US, 1972, pp. 85–103 (on pages 1, 9).

[Kha79] Leonid G. Khachiyan. “A polynomial algorithm in linear programming.” Russian. In: *Dokl. Akad. Nauk SSSR* 244 (1979), pp. 1093–1096 (on page 10).

[KV12] Bernhard Korte and Jens Vygen. *Combinatorial Optimization.* Springer Berlin Heidelberg, 2012 (on page 10).

[Mei18] Arne Meier. “Lecture: Komplexität von Algorithmen”. Leibniz Universität Hannover, 2018 (on page 3).

[Sch03] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Vol. A. Springer Berlin Heidelberg, 2003 (on page 12).

[STV17] Ola Svensson, Jakub Tarnawski, and László A. Végh. *A Constant-Factor Approximation Algorithm for the Asymmetric Traveling Salesman Problem.* 2017. arXiv: 1708.04215 [cs.DS] (on page 38).

[STV18] Ola Svensson, Jakub Tarnawski, and László A. Végh. “A constant-factor approximation algorithm for the asymmetric traveling salesman problem”. In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing - STOC 2018* (2018) (on pages 2, 4, 38).

[Sve15] Ola Svensson. “Approximating ATSP by Relaxing Connectivity”. In: *2015 IEEE 56th Annual Symposium on Foundations of Computer Science.* Oct. 2015, pp. 1–19 (on pages 1, 2, 18, 26, 38).

Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit selbstständig verfasst wurde, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden, alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind, und die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen hat.

Hannover, den 29.01.2020

Adrian Armstrong