KTH Computer Science
and Communication

On Approximating Asymmetric TSP and Related Problems

ANNA PALBOM

Licentiate Thesis
Stockholm, Sweden 2006

TRITA CSC-A 2006:4
ISSN 1653-5723 KTH School of Computer Science and Communication
ISRN KTH/CSC/A--06/04--SE SE-100 44 Stockholm
ISBN 91-7178-329-6 SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framläges till offentlig granskning för avläggande av filosofie licentiatexamen i datalogi fredagen den 19 maj 2006 klockan 14.00 i D3, Huvudbyggnaden, Kungl Tekniska högskolan, Lindstedtsvägen 5, Stockholm.

Tryck: Universitetsservice US AB

Abstract

In this thesis we study problems related to approximation of asymmetric TSP. First we give worst case examples for the famous algorithm due to Frieze, Gabiati and Maffioli for asymmetric TSP with triangle inequality. Some steps in the algorithm consist of arbitrary choices. To prove lower bounds, these choices need to be specified. We show a worst case performance with some deterministic assumptions on the algorithm and then prove an expected worst case performance for a randomised version of the algorithm. The algorithm by Frieze et al. produces a spanning cactus and makes a TSP tour by shortcuts. We have proven that determining if there is a spanning cactus in a general asymmetric graph is an **NP**-complete problem and that finding a minimum spanning cactus in a complete, directed graph with triangle inequality is equivalent to finding the TSP tour and the problems are equally hard to approximate. We also give three other results; we show a connection between asymmetric TSP and TSP in a bipartite graph, we show that it is **NP**-hard to find a cycle cover in a bipartite graph without cycles of length six or less and finally we present some results for a new problem with ordered points on the circle.

Sammanfattning

Denna licavhandling – vars titel kan översättas med *Om approximation av asymmetrisk TSP och besläktade problem* – tar upp några olika resultat som rör just approximation av TSP. För asymmetrisk TSP finns det en över tjugo år gammal approximationsalgoritm av Frieze, Gabiati och Maffioli [12]. Algoritmen bygger genom upprepade cykeltäckningar en uppspännande kaktus och bildar en TSP-tur genom att dra genvägar. Algoritmen ger en TSP-tur som är mindre än $\log_2 n$ gånger den kortaste turen. Papadimitrio och Vempala [27] har visat att om $\mathbf{P} \neq \mathbf{NP}$ så kan inte TSP-turen approximeras bättre än $220/219 - \epsilon$ gånger den kortaste turen. Eftersom den algoritmen och den undre gränsen är långt ifrån varandra kan någon av dem förbättras betydligt. Det finns två förbättringar av algoritmen. 2003 gav Bläser [5] en algoritm som ger en approximation i $0.999 \log_2 n$. Senare kom Kaplan, Lewenstein, Shafrir and Sviridenko [21] med en algoritm som ger en approximation inom $O(0.842 \log_2 n)$.

I Kapitel 2 ger vi värsta-fallet-grafer för algoritmen av Frieze m. fl. Först visar vi att med några deterministiska antaganden ger algoritmen en TSP-tur större än $\log_2 n \cdot OPT/(2 + o(1))$. Sedan visar vi att en slumpmässig version av algoritmen ger förväntad vikt i $\Omega(\log n)$ men med en sämre konstant.

Algoritmen av Frieze m. fl. bygger en uppspännande kaktus i grafen och bildar en TSP-tur genom att dra genvägar. I Kapitel 3 visar vi att avgöra om det finns en uppspännande kaktus i en generell graf är **NP**-fullständigt och att hitta en minimal uppspännande kaktus i en fullständig, viktad graf med triangelolikheten är ekvivalent med att hitta den minsta TSP-turen. Båda problemen är dessutom lika svåra att approximera.

Till slut tar vi upp tre andra resultat. Två rör bipartita grafer och det tredje ett nytt problem om ordnade punkter på en cirkel.

Contents

Chapter 1

Introduction

When solving a problem on a computer there are usually two limiting resources, time and space. How long time will it take and how much memory is needed? Since computers are getting faster and faster a more interesting question is how fast do these parameters grow with the size of the problem. If the size of the input of the problem is n, can the problem be solved with a number of operations which is linear in n? quadratic in n? or maybe exponential in n? In complexity theory problems are classified by this rate.

1.1 A Real-Life Problem

For example [10], a company is drilling n holes in a metal sheet. The drill has a starting position h_0 and moves from hole to hole. After all holes have been drilled the drill returns to the starting position. The distance between two holes is $d(h_i, h_j)$. Suppose that the time to move the drill is proportional to the distance. Drilling a hole takes time t_i. For an order π of the holes the total time required is $\sum_{i=0}^{n}(t_{\pi(i)} + k \cdot d(h_{\pi(i)}, h_{\pi(i+1)}))$ where $\pi(n+1) = \pi(0)$. The time required to drill, $\sum_{i=1}^{n} t_i$, is independent of π but we can change the order of the holes to minimise the moving time $k \sum_{i=0}^{n} d(h_{\pi(i)}, h_{\pi(i+1)})$.

If there are ten holes to be drilled then there are $10! = 3628800$ different orders and a computer can calculate the length of each order and select the shortest. But if there are one hundred holes, then there are $100! \approx 9 \cdot 10^{157}$ orders and it takes too much time to try each order.

1.2 Classification of Problems

The drilling problem is an *optimisation* problems where the best solution is wanted. The drilling problem is a hard problem where no polynomial time algorithm that finds the best solution is known. An optimisation problem has a corresponding *decision* problem. The decision problem corresponding to the drilling problem is

"Is there an order where the drill moves less than a distance k?". Characteristic for a decision problem is that the answer is yes or no.

Another optimisation problem is the shortest path problem "Find the shortest path from a to b in the graph G". For this problem there is a polynomial time algorithm which finds the best path. The corresponding decision problem is "Is there a path with length shorter than k between a and b in the graph?". Naturally also this problem can be solved in polynomial time. There are also decision problems without natural corresponding optimisation problems for example "Is the integer n an even square?". Also this problem can be solved in polynomial time.

A complexity class is a group of problems which are consider equally hard to solve. Usually the limiting resources is time or space. In this theses we will refer to the following two classes where time is the limiting resource; the class **P** is decision problems which can be solved in polynomial time or reasonable time, the class **NP** is a class of decision problems where if the answer is yes there is a proof that the answer is correct which can be checked in polynomial time. Of cause $\mathbf{P} \subseteq \mathbf{NP}$. A common belief is that $\mathbf{P} \neq \mathbf{NP}$ in other words that there are problems which have proofs that are easy to check but where the problem can not be solved in reasonable time. The drilling problem is one such problem. The proof that there is an order for which the drill moves less than k is the order it self. Given an order it is easy to check if the drill moves less than k, but finding the order is believed to be difficult. Many **NP** problems are proven to be equally hard to solve. If one can be solved in reasonable time so can the others. This group of connected problems are called **NP**-complete. Unfortunately many problems were early shown to be **NP**-complete [13]. The drilling problem is an **NP**-complete problem.

If it is difficult to decide if there is an order for which the drill moves less than a distance k of course it is difficult to find the best order. For an optimisation problem, if the corresponding decision problem is in **NP** a bit carelessly the optimisation problem is said to be in **NP**. Consider an optimisation problem in **NP** (where we can not find the optimal solution in reasonable time), can we find an approximate solution?

A minimisation approximation algorithm has a so-called approximation factor c if the solution is never worse than c times the optimum solution. The lower the value of c the better approximation algorithm and if $\mathbf{P} \neq \mathbf{NP}$ then $c > 1$. There is a way to decide if an approximation algorithm is optimal. By reducing another **NP**-complete problem to the problem one can decide a so-called *lower bound*, k, such that if $\mathbf{P} \neq \mathbf{NP}$ the approximation factor can never be lower than the lower bound, $c \geq k > 1$. If $c = k$ the approximation algorithm and the lower bound are optimal and there is no use trying to improve the algorithm (in a theoretical aspect, practical improvements can still be done). **NP**-complete problems have very different behaviour with respect to approximability. Some are possible to approximate within a constant arbitrarily close to one and others are not possible to approximate within any constant. In [3] there is a list of more than 200 **NP**-complete optimisation problems together with their approximability.

1.3 The Travelling Salesman Problem

The problem of drilling holes is an example of the Travelling Salesman Problem or TSP. As the name indicates the Travelling Salesman Problem considers a salesman who wishes to visit n cities in an order that makes the travelled distance as short as possible. He starts and ends in the same city and visits all other cities exactly once. It is also assumed that there is a road between every pair of cities. With this formulation the problem is not very realistic. There is not a road between every pair of cities and many other factors than the distance are also important for the choice of tour. But many other formulations of TSP, such as the drilling problem, are important problems to study.

TSP is an **NP**-complete problem and is hence not believed to be solvable in reasonable time. Therefore much effort has been put into finding approximation algorithms. When studying the problem formally the cities are nodes in a complete graph with positive weights on the edges and the problem is to find the shortest Hamiltonian cycle.

TSP was proven to be **NP**-complete already by Karp [22] in 1972, and Sahni and Gonzalez [30] showed that it is **NP**-complete to find a tour with length within exponential factors of the optimum. When the distance function is constrained to satisfy the triangle inequality, $d(v_x, v_y) \leq d(v_x, v_z) + d(v_z, v_y)$, the best known approximation algorithm has an approximation factor $c = 3/2$ and is due to Christofides [7]. The algorithm finds the minimum spanning tree in the graph and then makes a minimum cost matching of nodes with odd degree. The result is an Eulerian graph. The TSP tour is made by shortcuts in the Eulerian graph. The weight of the tree is less than the weight of the TSP tour and the matching has weight at most half the weight of the TSP tour. Since the graph obeys the triangle inequality the shortcuts do not increase the weight. Hence the result is at most 3/2 times the optimum solution.

Practical Algorithms

The distances in the drilling problem do not only obey the triangle inequality, they are also Euclidian distances. For many real world problems this is the case and therefore many heuristics have been invented to solve or approximate them. Currently the best tool to find the optimal solution to large-scale Euclidian instances of TSP is the Concorde [1]. There has been a challenge to find the best TSP tour visiting all 24 978 cities in Sweden (using Euclidian distances). In May 2004 the challenge was won by the program Concorde which solved the problem with linear programming. It was also proven that the solution was optimal. An Euclidian TSP instance with up to 1000 nodes is solved exactly within some minutes with the Concorde.

When it comes to approximation there are many different algorithms. The performance of an algorithm depends on the graph and on the implementation, so it is difficult to compare them. If one wants to choose approximation algorithm

for an instance of TSP, the best way is to test the algorithms in question on the particular instance. The rule of thumb is what you win in time you loose in quality. Johnson et al. [19] compare a variety of algorithms for different graphs but focus on 2-dimensional Euclidian distance functions. When we refer to their investigation we will give the performance time for 10 000 nodes uniformly distributed in the plane. The algorithms were run on a Compaq ES40 with 55-MHz Alpha processors.

One of the fastest algorithms divides the plane into strips, sorts the nodes in each strip and then connects them. In 0.01 seconds it gives an approximation 1.31 times the optimal solution. The algorithm nearest neighbour chooses in every step the nearest neighbour as the next city. The algorithm is fast but might perform poorly, in 0.28 seconds it gives a tour 1.25 times the optimum. First producing a tour with for example nearest-neighbour and then replacing edges by different local searches are the algorithms which give the best approximations. 3-Opt makes a TSP tour with nearest neighbour and then tries improve the tour by deleting three edges and permute the three resulting paths. It produces an 1.03-approximation in 1.5 seconds. The famous Lin-Kernighan has a more intricate way of changing edges and is more effective than 3-Opt . It gives an 1.02 approximation in 2.3 seconds. (For larger testbeds the difference in the approximation constant is larger.) Helsgaun is a complex heuristic and gives a very good approximation. The disadvantage is that it is rather slow. In 862 seconds it gives a 1.008-approximation.

Johnson et al. also present results for the algorithm by Christofides. Even though it is the only algorithm with a proven upper bound it is far from optimal in practice. In 1.04 seconds it returns an 1.1-approximation.

1.4 Asymmetric TSP

Another drilling problem is when the metal sheet is tilted. The drill is first moved vertically, y, and then horizontally. The speed moving the drill up y^+ is faster than the speed moving down y^-. This is also TSP but the distance function is *asymmetric*.

Formally the distance matrix is not restricted to being symmetric. This case is much less understood than symmetric TSP. In 1982 Frieze, Galbiati and Maffioli [12] invented their famous algorithm for *asymmetric* graphs with triangle inequality, which approximates the TSP tour within a factor of $\log_2 n$. The main idea of the algorithm by Freize et al. is: Find a minimum cycle cover in the graph by linear programming. Choose one node in every cycle and form a subgraph with the same distance function as in the original graph. Find a minimum cycle cover in the subgraph. Continue iteratively until there is only one cycle in the cycle cover. Similar to the algorithm by Christofides the union of the cycle covers form an Eulerian graph. Replace edges in the union with a shortcut edge to obtain a TSP tour. Since the graph respects the triangle inequality the TSP tour has weight less than or equal to the the sum of the cycle covers.

There is only a small lower bound: Papadimitriou and Vempala [27] recently proved that it is **NP**-hard to approximate the minimum TSP tour within a factor of $220/219 - \epsilon$, for any constant $\epsilon > 0$. Obviously, huge improvements can be made either on a better algorithm or a tighter lower bound. Despite a lot of effort during the last twenty years there have been only two algorithmic improvements, both very recent. The first by Bläser was announced in 2003 [5]. He improves the algorithm by Frieze et al. and proves a factor $0.999 \cdot \log_2 n$. The second algorithm by Kaplan, Lewenstein, Shafrir and Sviridenko [21] decomposes multigraphs and gives an approximation of $3/4 \log_3 n < 0.842 \log_2 n$.

Practical Algorithms

Practical heuristics for approximation are not as well developed as for the symmetrical case. Johnson et al. [18] have made a comparison of most algorithms on different testbeds. There is no obvious choice of testbed and the algorithms perform differently depending on the data. Generally the approximation algorithms are slow, the largest testbed has 3162 nodes, and the approximation factors are larger than in the symmetric case. Optimisation is also slow and for several testbeds with 1000 nodes the optimal tour was never found. They had drilling on a tilted sheet as one testbed (this distances obey the triangle inequality) and when we refer to there investigation we use performance times for the testbed with 1000 drillholes on a tilted sheet.

There are not as many algorithms developed for asymmetric TSP as for symmetric. Many algorithms for symmetric TSP can not handle asymmetric distances. The distances are not Euclidian, so it is not possible to divide the nodes into strips with respect to position. Local search algorithms where small local optimal paths are connected have problems since the short paths can be in opposite directions and are hence expensive to connect. One algorithm that can handle asymmetric distances is nearest neighbour. It gives an 1.28-approximation in 1.9 seconds. Another algorithm that also works for asymmetric graphs is 3-Opt in 6.6 seconds 3-Opt gives an 1.18-approximation.

One can construct a symmetric counterpart to an asymmetric graph. Replace each node v_i by v_i^+ and v_i^-. An edge (v_j, v_i) is replaced by (v_j^-, v_i^+) were $d(v_j^-, v_i^+) = d(v_j, v_i)$ and the edge (v_i, v_j) is replaced by (v_i^-, v_j^+) where $d(v_i^-, v_j^+) = d(v_i, v_j)$. Let $d(v_i^+, v_i^-) = -\infty$ and all other edges has weight ∞ (implemented as a large number). This construction might cause problems because of the large edge weights, nonpositive weights, loss of triangle inequality and also increases the number of nodes but it is solvable with some algorithms for symmetric TSP. Making the graph symmetric and then use the algorithm Helsgaun gives in 318 seconds an 1.01-approximation.

The algorithm by Frieze et al. was also tested. Even though it is the only algorithm with a proven upper bound, an algorithm which builds only *one* cycle cover and then patches the cycles together was both faster and gave a better approxim-

ation for all testbeds. In 2.9 seconds the algorithm with one cycle cover gave an 1.18-approximation.

1.5 Variants of TSP

Many other variants of TSP have been studied. In Max-TSP or informally the "taxicab ripoff problem" the maximum TSP tour is sought. If the edges are allowed to have negative weights it is the same problem as minimum TSP. In most cases though only non-negative edge weights are allowed, in which case it is easier to approximate than minimum TSP and there is an approximation algorithm by Kosaraju et al. [24] which gives a TSP tour of length at least 38/63 times the maximum tour.

Group-TSP or TSP with neighbourhood is when a salesman wants to visit n customer and the customer can move within a neighbourhood. Safra and Schwartz [29] give some lower bounds for Group-TSP with restricted distances.

Price collecting TSP is when a visit to a city gives a price reward. The goal can be to collect a certain amount of money and to minimise the travelled distance. Or there is a limitation of the distance and the goal is to maximise the collected money. The problems have in common that the goal is to choose a subset of the nodes and to find an certain ordering of the selected nodes. Balas gives in [4] a summary over problems and results in this category.

TSP with multiple salesmen or k-TSP is the problem when k salesmen shall visit n cities, all staring and ending in the same city [28].

In practice there can be time restrictions. A city must be visited within a certain time interval [2], or the sheet where holes are drilled is moving so the distances depend on time [14].

1.6 Results in the Thesis

Most results concern asymmetric TSP in one way or another. Chapter 2 shows a worst case performance of the approximation algorithm of asymmetric TSP with triangle inequality by Frieze et al. [12]. Their analysis of the algorithm gives a upper bound. We construct a family of worst case graphs for the algorithm with the following assumptions on the algorithm:

1. The *first* node in every cycle is chosen for the subgraph.
2. The shortcuts are made in a certain specific order.

We show that the analysis is tight up to a constant factor. This section is based on the paper [25]. Then we construct a family of graphs which obeys an expected worst case performance with the following weaker assumptions:

1. A *random* node in every cycle is chosen for the subgraph.
2. The shortcuts are made by a depth-first search.

but get a worse constant.

Both the algorithm by Christofides for symmetric TSP and the algorithm by Frieze et al. for asymmetric graphs, find an Eulerian graph and produce a TSP tour by shortcuts. In an asymmetric graph one Eulerian subgraph is the spanning cactus. Is it possible to find a spanning cactus with less weight than the one produced by Frieze et al.? Chapter 3 is based on the article [26] and shows that determine if there is a spanning cactus in a general asymmetric graph is **NP**-complete and that finding the minimum spanning cactus in a complete graph is polynomial time equivalent to TSP and they have the same hardness in approximation.

In Chapter 4 we investigate some correlations between TSP in an asymmetric graph and TSP in a bipartite graph. This work was done together with Lars Engebretsen and my contribution to the results in this chapter is approximately 50%.

In [15] Hartvigsen claim that finding a perfect 2-matching without 6-factors in a bipartite graph is **NP**-complete. To our knowledge, no proof has been published. We give a proof in Chapter 5.

A new problem is the Betweenness problem on the circle. In Chapter 6 we show that it is **NP**-complete and give an approximation algorithm.

1.7 Acknowledgement

The work leading to this thesis has been carried out at the Theoretical Computer Science group at CSC, KTH. I am grateful to all members of the group making it a stimulating and pleasant environment to work in. I want to thank my supervisor Mikael Goldman for great support during my years as PhD student, my assistant supervisor Lars Engebretsen for many interesting discussions and Johan Håstad for encouraging leadership of the group. Lena Folke and Simon Wigzell have helped with correcting my English, thank you!

A big thank yo my friends and family for being such nice friends and family. I am also grateful to my grandparents who took the first steps on our journey into academics, to Jan for being who you are and letting me be who I am and for Alma who brings joy to my life.

Chapter 2

Worst Case Performance of the Approximation Algorithm by Frieze et al. for Asymmetric TSP

In 1982 Frieze, Galbiati and Maffioli (Networks 12:23-39) published their famous algorithm for approximating the TSP tour in an *asymmetric* graph with triangle inequality. They show that the algorithm approximates the TSP tour within a factor of $\log_2 n$. We construct a family of graphs for which the algorithm (with some implementation details specified by us) gives an approximation which is $\log_2 n/(2+o(1))$ times the optimum solution. We also relax the assumptions of the algorithm and show an expected worst case preformance, but the constant is worse in this case. This shows that the analysis by Frieze et al. is tight up to a constant factor and can hopefully give deeper understanding of the problem and new ideas in developing an improved approximation algorithm.

2.1 Introduction

The Travelling Salesman Problem (TSP) is one of the most famous and well-studied combinatorial optimisation problems.

Definition 1 *The* (Asymmetric) TSP *is the following minimisation problem: Given a collection of cities and a matrix whose entries are interpreted as the non-negative distance from a city to another, find the shortest tour starting and ending in the same city and visiting every city exactly once.*

TSP was proven to be **NP**-hard already by Karp [22] in 1972. This means that an efficient algorithm for TSP is highly unlikely; hence it is interesting to investigate algorithms that compute *approximate* solutions. However Sahni and Gonzalez [30] showed that in the case of general distance functions it is **NP**-hard to find a tour with length within exponential factors of the optimum, this is true even if the

graph is restricted to being symmetric. When the distance function is symmetric and constrained to satisfy the triangle inequality the best known approximation algorithm is a factor 3/2-approximation algorithm due to Christofides [7]. With a c-approximation algorithm we mean a polynomial time algorithm that outputs a tour with weight at most c times the optimum weight.

We will study the case when the distance function satisfies the triangle inequality but is not limited to being symmetric. This case is much less understood. In 1982 Frieze, Galbiati and Maffioli [12] invented their famous algorithm for *asymmetric* graphs with triangle inequality, which approximates the TSP tour within a factor of $\log_2 n$. There is only a small lower bound: Papadimitriou and Vempala [27] recently proved that it is **NP**-hard to approximate the minimum TSP tour within a factor less than $220/219 - \epsilon$, for any constant $\epsilon > 0$. Obviously, huge improvements can be done either on a better algorithm or a tighter lower bound. Despite a lot of effort during the last twenty years there have only been two algorithmic improvements, both very recent. The first by Bläser was announced in 2003 [5]. He improves the algorithm by Frieze et al. and proves a factor $0.999 \cdot \log_2 n$. The second algorithm by Kaplan, Lewenstein, Shafrir and Sviridenko [21] decomposes multigraphs and gives an approximation of $3/4 \log_3 n < 0.842 \log_2 n$. Hence, any new insight regarding the asymmetric TSP is important. One way to achieve such insight is to identify potential "hard" instances for the known approximation algorithms. The algorithms by Bläser and by Kaplan et al. are more complicated than the original algorithm due to Frieze et al. and are hence more difficult to understand. Therefore, we study the original algorithm in this paper. By constructing an explicit family of graphs we establish that the analysis of the algorithm is tight up to a constant factor (Theorem 1 and 2).

The main idea of the algorithm by Freize et al. is: Find a minimum cycle cover in the graph by linear programming. Choose one node in every cycle and form a subgraph with the same distance function as in the original graph. Find a minimum cycle cover in the subgraph. Continue iteratively until there is only one cycle in the cycle cover. The union of the cycle covers form an Eulerian graph. Replace edges in the union with a shortcut edge to obtain a TSP tour. Since the graph respects the triangle inequality the TSP tour has weight less than or equal to the the sum of the cycle covers. First we construct a family $\mathcal{H}$ of graphs which have a worst case performance of the algorithm by Frieze et al. with the following assumptions on the algorithm;

1. The *first* node in every cycle is chosen for the subgraph.
2. The shortcuts are made in a certain specific order.

This shows that the analysis of the algorithm by Frieze et al. is tight and our first main result is:

Theorem 1 *For every ϵ $(0 < \epsilon < 1/n)$, there exists a family $\mathcal{G}$ of graphs, G_n, such that the approximation algorithm by Frieze et al. [12] with our deterministic*

specifications, gives a TSP tour, T such that

$$\frac{T}{opt(G_n)} > \frac{\log_2 n}{2 + 2\epsilon}$$

Then we have constructed a family, $\mathcal{G}$, of graphs which obeys an expected worst case performance with the following weaker assumptions:

1. A random node in every cycle is chosen for the subgraph.
2. The shortcuts are made by a depth-first search.

With these assumptions the expected weight of the TSP tour is proportional to $\log n \cdot opt(G_n)$ but the constant is worse than before. The spanning cactus is heavy regardless of the choice of node in the subgraph. The first assumption is needed for the expected length of the TSP tour. The second assumption that the shortcuts are made by a depth-first search is a fast and natural implementation and is also needed for the lower bound of the length of the TSP tour.

Theorem 2 *There exists a family, $\mathcal{G}$, of graphs, G_n, such that the approximation algorithm by Frieze et al. [12] with random choices gives a TSP tour with expected weight*

$$\Omega(opt(G_n) \log n)$$

Both constructions are symmetric graphs and they can easily be approximated by the algorithm by Christofides. For the first construction we give as a corollary a family of asymmetric graphs which also have weight respective expected weight of the TSP tour proportional to $opt(G_n^a) \log n$. For asymmetric graphs Frieze et al. [12] give another data dependent algorithm which gives a $3\alpha/2$-approximation of the TSP tour, where α is the maximum ratio of $d(v_i, v_j)/d(v_j, v_i)$ for $v_i, v_j \in V$, $v_i \neq v_j$. The idea of the algorithm is to make the graph symmetric and then use the algorithm for symmetric graphs by Christofides [7]. We will not show a worst case behaviour of this algorithm, but the family of asymmetric graphs is not guaranteed to be well-approximated by the data dependent algorithm.

Some terminology

All graphs in this chapter have $n = 2^m$ nodes placed in a circle. When an algorithm operates on an arbitrary node the ordering is modulo n. For example the node before v_0 is v_{n-1} and $v_0 = v_n$. An interval of node-indexes $[a, b]$ represents, if $a < b$ all numbers $a \leq i \leq b$ and if $b < a$ the numbers $[a, n-1] \cup [0, b]$. The length of an interval is if $a < b$ $d[v_a, v_b] = \sum_{i=a}^{i<b} d(v_i, v_{i+1})$ and if $a > b$ $d[v_a, v_b] = \sum_{i=a}^{i<n} d(v_i, v_{i+1}) + \sum_{i=0}^{i<b} d(v_i, v_{i+1})$.

We often discuss parts of graphs; we therefore introduce some terms describing such parts. By cycle we mean simple cycle and a cycle is denoted by the nodes in it written in the cycle order. For example $c = (v_1, v_2, v_3)$ is the directed cycle from

v_1 to v_2 to v_3 and back to v_3. Sometimes we do not know which index the nodes have on the circle, then a cycle i is denoted $C_i = (v_{i_1}, v_{i_2}, v_{i_3})$ and i_j is a symbol for the index on the circle. The weight of a cycle $w(C) = \sum_{e \in C} d(e)$ is the sum of the weight of the edges in the cycle.

Definition 2 *A* directed cactus *is a strongly connected, asymmetric graph where each edge is contained in at most (and thus, in exactly) one simple directed cycle* [31]. *A* spanning cactus *for an asymmetric graph G is a subgraph of G that is a directed cactus and connects all vertices in G.*

The weight of a cactus $w(K) = \sum_{e \in K} d(e)$ is the sum of the weight of the edges in the cactus. Throughout the paper, T is a TSP tour, $opt(G)$ is the weight of the minimum TSP tour in the graph G, and C is a cycle, K is a cactus and $\mathcal{K}$ is a set of cacti.

Given a strongly connected graph, $G = (V, E)$ with weighted edges, we define the distance between two nodes, $d(v_i, v_j)$, as the weight of the shortest path in G from v_i to v_j. This distance function clearly obeys the triangle inequality.

2.2 The Approximation Algorithm

An intuitive description of the approximation algorithm by Frieze et al. [12] is given in the introduction. Their description of the algorithm, is given in this section.

The main algorithm is ATSP; it calls the procedure ASSIGN which returns a minimum cycle cover and the procedure TOUR which makes the shortcuts.

Procedure ASSIGN(G, D)

Input: A graph $G = (V, E)$
A cost function $D : E \to Q^+$
Output: A cycle cover $\mathcal{C} \subset E$
The procedure finds a set $\mathcal{C} \subset E$ of minimum cost such that every node in V has in and out degree equal to one.

For the next procedure we need some notations: In a spanning cactus K the following holds for every node $v \in V$:

1. $indegree(v) = outdegree(v) = deg(v)$

2. the removal of node v from K leaves $deg(v)$ connected components

Remark 1 *Frieze et al. make the observation that these properties imply that for each connected component K_i obtained by removing v there exists nodes $u_i, w_i \in K_i$ such that (u_i, v) and (v, w_i) are in K.*

Procedure TOUR(G, K)

Input: A graph $G = (V, E)$
A spanning cactus $K \subset E$
Output: A TSP tour K
begin
while there exists a node $v \in V$ with $deg(v) > 1$ do
begin
$K \leftarrow K \cup (u_1, w_2)$*;
$K \leftarrow K \setminus \{(u_1, v), (v, w_2)\}$;
end
end

* Remark 1

Procedure ATSP(G_0, D_0)

Input: A graph $G_0 = (V, E)$
A cost function $D_0 : E \rightarrow Q^+$
Output: A TSP tour T
begin
1 $K \leftarrow \emptyset$;
2 $D \leftarrow D_0$;
3 $G \leftarrow G_0$;
4 $k \leftarrow 2$;
5 while $k \neq 1$ do
6 begin
7 $\{C_1, C_2, ..., C_h\} \leftarrow ASSIGN(G, D)$;
8 $V \leftarrow \emptyset$;
9 for $i = 1$ until h do
10 begin
11 choose* a node v_i of C_i;
12 $V \leftarrow V \cup \{v_i\}$;
13 $K \leftarrow K \cup \{C_i\}$;
14 end
15 Let G be the complete subgraph of G_0 induced by V and
D the induced cost matrix of G;
16 $k \leftarrow h$;
17 end
18 $T \leftarrow TOUR(G_0, K)$;
19 return T;
end

In the last iteration the single node in V is the *root* of the cactus. To get an upper bound on their algorithm Frieze et al. make the following analysis: In the

worst case all cycles in the cycle cover have length two, hence at most $\lfloor \log_2 n \rfloor$ cycle covers are produced. The weight of every cycle cover is less than or equal to $opt(G)$. Thus the spanning cactus formed by the union of the cycle covers has weight at most $opt(G) \cdot \log_2 n$. Since the graph obeys the triangle inequality the TSP tour found from the spanning cactus is shorter than or equal to $opt(G) \cdot \log_2 n$.

2.3 Deterministic Assumptions on the Algorithm

Specification of the Approximation Algorithm

In order to analyse the algorithm we need to specify the arbitrary choices in the description by Frieze et al.:

1. An arbitrary node from every cycle in a cycle cover is chosen to be in the next subgraph (row 11 in DTSP). We choose the node with lowest index.
2. The shortcuts made to transform the spanning cactus to a TSP tour in the procedure TOUR are in arbitrary order. We use our procedure SHORTCUT below which makes the shortcuts in a specific order.

SHORTCUT(G, K, r)
Input: A graph $G = (V, E)$
A cactus $K \subset E$
The root node r

Output: A TSP tour P
begin
global $P \leftarrow \emptyset$;
global set of visited nodes $U \leftarrow \emptyset$;
$t \leftarrow$ DEPTH-FIRST (G, K, r, r);
$P \leftarrow P \cup (t, r)$;
return P;
end

DEPTH-FIRST(G, K, r, s)
Input: A graph $G = (V, E)$
A cactus $K \subset E$
A root node r
The present node s
Output: The last node added
begin
$t \leftarrow s$, $U \leftarrow U \cup \{s\}$;
for all* $(s, v) \in K : v \notin U$ do
$P \leftarrow P \cup (t, v)$;
$t \leftarrow$ DEPTH-FIRST(G, K, r, v);
end
return t;
end

* The edges $e = (s, v_i) \in K$ are selected in decreasing order first with respect to $l_r(e)$ and then with respect to index i.

The algorithm with these specifications is called DTSPS.

Notations and conventions

Definition 3 *For an m-bit integer x the we define the function*

$$z_m(x) = \max\{k \in Z_m \mid 2^k \textit{ divides } x\}$$

In particular $z_m(0) = m - 1$ since all numbers divide zero.

In words, $z_m(i)$ is, for $i \neq 0$, the position of the least significant non-zero bit in the binary representation of i.

Definition 4 *A* cycle cover *for a directed graph $G = (V, E)$ is a subgraph of G such that for each node $v \in V$, $indegree(v) = outdegree(v) = 1$. A cycle cover where every cycle has exactly two nodes is called a* 2-cover.

Definition 5 *Given a directed cactus K and a root node r we define for a simple cycle $C \in K$ a function $l_r(C)$ in the following way: If $r \in C$ then $l_r(C) = 1$. The rest of the function values are defined iteratively. For a cycle C with $l_r(C) = j$ a cycle C' without function value, connected to C gets $l_r(C') = j+1$. Since the graph is a cactus the function value is unique. For every edge e in the cycle C we define with a slight abuse of notation $l_r(e) = l_r(C)$. For a node in the graph we define*

$$l_r(r) = 0$$
$$l_r(s) = \min\{l_r(e) | e \text{ incident to } s\}$$

The level function defines a partial order of the cycles in the cactus.

Definition 6 *For a cactus K, a root node r and an edge e, let C_e be the cycle containing e and the subcactus $B_{K,r,e}$ be the the connected component containing e in*

$$C_e \cup \Big(\bigcup_{l_r(C) > l_r(C_e)} C \Big)$$

The union of all such subcacti starting in a node s is defined by

$$B_{K,r,s} = \bigcup_{\substack{e \text{ incident to } s \\ l_r(e) > l_r(s)}} B_{K,r,e}$$

The complete graph with nodes in the subcactus $B_{K,r,s}$ is denoted $\mathcal{K}_{K,r,s}$.

The last definition is analogus to a *subtree*.

Analysis of SHORTCUT

A straightforward analysis shows that the tour produced by the procedure SHORTCUT is a TSP tour. Since the algorithm is independent of the distance function this TSP tour only depends on the structure of the spanning cactus. A comparison with the original algorithm by Frieze et al. shows the following:

Lemma 1 *The TSP tour produced by SHORTCUT on a spanning cactus can be produced by the algorithm by Frieze et al. on the same graph.*

Proof 1 *In the algorithm by Frieze et al. the procedure TOUR transforms the spanning cactus into a TSP tour. We prove that SHORTCUT can be simulated by TOUR.*

Let K be the initial cactus. The procedure SHORTCUT repeatedly adds edges to a TSP tour P. TOUR stepwise adds and removes edges to the cactus. When simulating SHORTCUT with TOUR let T denote the edges in the cactus at the current step. Initially $T = K$ and $P = \emptyset$ and our task is to prove that in the end $T = P$ with a certain order of the shortcuts in TOUR.

Claim: Given a complete graph G, a spanning cactus K with root in node r DEPTH-FIRST(G, K, r, r) repeatedly adds nodes to a simple path P starting in r and returns the last node t in the path. The path P connects all nodes in the graph. This process can be simulated by TOUR in such a way that if $K = T$ before any step in TOUR, the order of the shortcuts can be chosen such that the set T, when the simulation is finished, has the following properties: $(t, r) \notin P$ and $P \cup (t, r) = T$.

If the claim is true we get the following: The procedure SHORTCUT calls DEPTH-FIRST with start node r. Then $K = T$ and $P = \emptyset$. By the claim DEPTH-FIRST adds nodes to P which is a simple path connecting all nodes in K, starting in r and ending in t. The order of shortcuts in TOUR can be chosen such that $T = P \cup (t, r)$. In the next step in SHORTCUT $P \leftarrow P \cup (t, r)$ and P is the final TSP cycle. Now $P = T$ and both procedures have returned the same TSP tour.

The proof of the claim is by induction on the number of cycles in the cactus. Figure 2.1 shows some of the cases we have to cover in the proof. For a cactus with one cycle $T = K = S = (r, v_1, ..., v_k)$ DEPTH-FIRST adds all but the last edge to P and returns $t = v_k$. Since $T = P \cup (v_k, r)$ the claim holds.

Assume that the claim is true for every cactus with less than n cycles and consider what happens when K in the claim is a cactus with n cycles: If there are several cycles connected to r all edges $e_i = (r, v_i), i = 1, ..., k$ have the same l_r.

Calling DEPTH-FIRST(G, K, r, r) is the same as calling DEPTH-FIRST($G \cap \mathcal{K}_{K,r,e_i}, B_{K,r,e_i}, r, r$) for all i and gluing the paths together. Since K is a cactus every B_{K,r,e_i} is a cactus with less than n cycles not connected to any other part of the graph. By assumption DEPTH-FIRST connects every node in $\mathcal{K}_{K,r,e_i}$ with a path P_i starting in r and returns a last node t. The shortcuts in TOUR can by assumption be chosen such that $T \cap \mathcal{K}_{K,r,e_i} = (P \cup (t, r)) \cap \mathcal{K}_{K,r,e_i}$. The same holds for every subcactus B_{K,r,e_i}. All paths P_i can be glued together such that $P = (\cup_i P_i \setminus \cup_{i>1}(r, v_i)) \cup_{i<k} (t_i, v_{i+1})$. For $1 \leq i < k$ $(r, v_{i+1}) \in T$ and $(t_i, r) \in T$ therefore TOUR can replace $(t_i, r), (r, v_{i+1})$ with (t, v_{i+1}) in T and the claim is true for this cactus as well.

If the root r has in- and out-degree one it is in a directed cycle $S = (r, v_1, ..., v_k)$ and $T = K$ (Figure 2.1); in particular $(v_k, r) \in T$. If a node $v_i \in S$ is not connected to any other cycle than S, DEPTH-FIRST adds the edge (v_i, v_{i+1}) to P. Since the edge is in T no shortcuts are made by TOUR when simulating DEPTH-FIRST. If a node v_i is connected to other cycles than S, calling DEPTH-FIRST(G, K, r, v_i) is the same thing as calling DEPTH-FIRST($G \cap \mathcal{K}_{K,r,v_i}, B_{K,r,v_i}, v_i, v_i$), gluing the paths together and then continue with the edge in the cycle S. This is true since;

K is a cactus and therefore the subcactus B_{K,r,v_i} is a cactus with less than n cycles not connected to any other part of the graph. The relative order of the cycles in this subcactus is the same if v_i or r is the root. DEPTH-FIRST chooses edges with respect to $l_r(e)$ and therefore the edges in the subcactus B_{K,r,v_i} are chosen first. By assumption DEPTH-FIRST connects all nodes in $\mathcal{K}_{K,r,v_i}$ with a path starting in v_i and ending in t. By the same assumption TOUR can add and remove edges in T such that $(P \cup (t, v_i)) \cap \mathcal{K}_{K,r,v_i} = T \cap \mathcal{K}_{K,r,v_i}$. When DEPTH-FIRST continue with the edge $(v_i, v_{i+1}) \in S$ $P \leftarrow P \cup (t, v_{i+1})$. Since $(v_i, v_{i+1}) \in T$ TOUR can perform $T \leftarrow (T \setminus \{(t, v_i), (v_i, v_{i+1})\}) \cup (t, v_{i+1})$. At the last node v_k in the circle S the recursion in DEPTH-FIRST halts since the next node r is visited. If v_k was not connected to any other cycle than S, $t = v_k$ and the claim holds. If v_k was connected to other cycles there is an edge $(t, v_k) \in T$ and TOUR can replace $\{(t, v_k), (v_k, r)\}$ with (t, r) in T and the claim also holds.

Preliminary construction

In this section we construct a simple family of graphs to illustrate the algorithm by Frieze et al. and the main ideas of the worst case performance. Most proofs are omitted and the purpose of the example is to give an intuitive understanding. Since the graphs are symmetric they can be approximated within 3/2 by the algorithm due to Christofides [7].

Constructing the graph

The distance function is induced by a graph (Fig. 2.2a) defined as follows:

Definition 7 *The distance function, $d_n^1(v_i, v_j)$, is induced by an undirected graph with $n = 2^m$ nodes arranged in a circle. Adjacent nodes are connected by edges of weight one and there are no other edges in the graph.*

Definition 8 *Let G_n^1 be a complete, directed graph with $n = 2^m$ nodes and the distance function $d_n^1(v_i, v_j)$.*

The distance between two nodes in G_n^1 is the difference in index; $d_n^1(v_i, v_j) = \min\{|i - j|, n - |i - j|\}$. The edges are directed even though they have the same distance in both directions. The minimum TSP tour is of course to traverse the nodes in clock-wise order (or counter-clock-wise) and $opt(G_n^1) = n$.

The spanning cactus

The algorithm by Frieze et al. recursively finds a minimum cycle cover in the graph. In a complete, asymmetric graph, the union of all cycle covers recursively produced by the algorithm forms a *spanning cactus*.

To get an intuitive understanding of the algorithm DTSPS and the worst case behaviour of G_n^1 we use a graph with $n = 2^4 = 16$ nodes as an example (Fig. 2.2a).

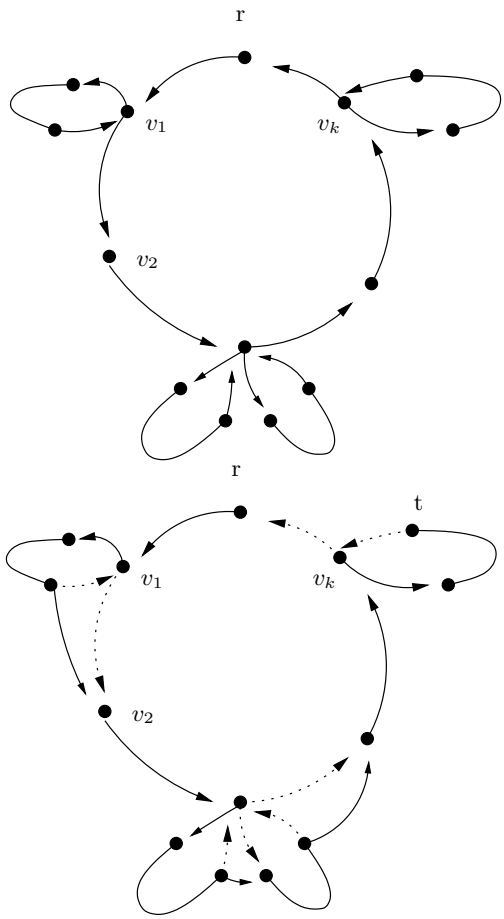

Figure 2.1: **Left (a):** A cactus with root r, the ellipsis symbolize parts of the directed cactus. **Right (b):** After simulating DEPTH-FIRST: Solid edges are edges added to the simple path P and t is the last node in P.

In the first recursion the minimum cycle cover can consist of one large cycle or eight of weight two. Both have total weight 16. Assume that the 2-cover is chosen. Choose the first node in every cycle to be in the set of nodes for the next recursion. In our example this gives the nodes with even index. Now the shortest distance between any nodes in the subgraph G is two. Again the procedure can return one large cycle or four cycles of weight four. Both have the total weight 16 and we assume that the 2-cover is returned. Proceed in the same way until there is just one cycle in the cycle cover. The union of all 2-covers is called a *W-cactus*.

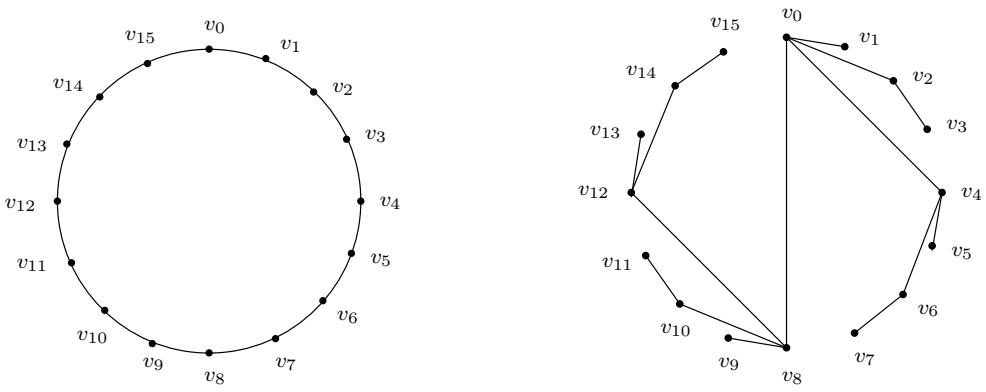

Figure 2.2: **Left (a)**: The graph inducing the distance function, $d^1_{16}(v_i, v_j)$. All edges have weight one. **Right (b)**: The Worst Case Spanning Cactus (*W-cactus*) in the graph G^1_{16}. Each 2-cycle is symbolised with an edge.

Definition 9 *For a graph G with $n = 2^m$ nodes a* Worst Case Spanning Cactus *or a* W-cactus *is a subgraph of G such that $E = \{(v_i, v_{i-2^k}), (v_{i-2^k}, v_i) : z_m(i) = k\}$.*

For $n = 16$ nodes the W-cactus looks like in Fig. 2.2b. It can be seen that a *W-cactus* in G^1_n has weight $n \log n$.

Lemma 2 *For a W-cactus, K, in a graph G with $n = 2^m$ nodes and an node $s = v_i$ the procedure DEPTH-FIRST returns v_i if i is odd and v_{i+1} if i is even.*

Proof 2 *Every node v_i in a W-cactus with odd index, i.e., such that $z_m(i) = 0$, is by Definition 9 in exactly one cycle (v_i, v_{i-1}). Every node v_i in a W-cactus with even index, i.e., such that $z_m(i) = k \geq 1$, is in least two cycles (v_i, v_{i+1}) and (v_i, v_{i-2^k}). When DEPTH-FIRST is called with an odd node $s = v_i$ as input there is no other cycle $(s, v) \in K$ and the procedure returns $t = s$. If the input node $s = v_i$ is even there may be several cycles $(s, v) \in K$ but the one with smallest difference in index is (v_i, v_{i+1}) and it is the last cycle selected in the loop. The node $v_{i+1} \notin U$ since there is just one cycle connecting the odd node v_{i+1} with the rest of the cactus. After the recursive call to DEPTH-FIRST $t \leftarrow v_{i+1}$. Hence the procedure returns $t = v_{i+1}$ in this case.*

To make the notation in some proofs clear we need the following definition:

Definition 10 *For the algorithm DTSPS and the graph $G_n = G_{n,0}$ with n nodes, the subgraph remaining after the first cycle cover is found is denoted by $G_{n,1}$ and the subgraph remaining after the i:th recursion is denoted by $G_{n,i}$.*

Since every cycle in the cycle cover is a 2-cycle of edges with equal weight, we visualize every 2-cycle as an undirected weighted edge. The union of the cycle covers can with this view be seen as a spanning, undirected tree. Since every node is in at least one cycle the tree is spanning and by the construction the cover is cycle-free. The view of the spanning cactus as a spanning tree directly gives that a W-cactus has $n - 1$ cycles.

The TSP tour

We proceed by analysing SHORTCUT. The procedure takes a spanning cactus and returnes a TSP tour. It is independent of the distance function and only considers the structure of the spanning cactus. Again we use the graph G_{16}^1 as an example to describe the procedure and the spanning cactus is a W-cactus. The procedure starts at node v_0. After some recursive calls to DEPTH-FIRST the graph will look like Fig. 2.3(a).

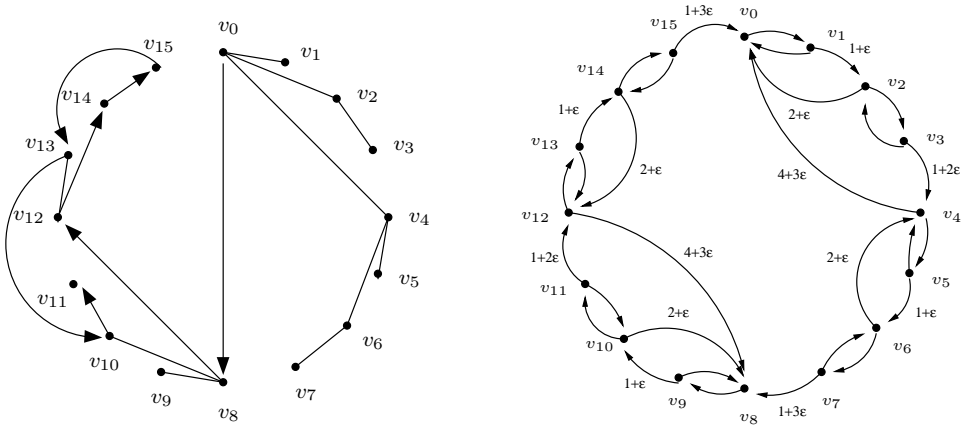

Figure 2.3: **Left (a)**: The TSP tour after some steps with the procedure SHORTCUT on a W-cactus with $n = 2^4 = 16$ nodes. Undirected edges represent 2-cycles in the W-cactus and arrows represent edges in the TSP tour. **Right (b)**: The graph inducing the distance function $d_{16}(v_i, v_j)$. For simplicity weights equal to one are omitted.

It can be seen that the TSP tour produced by DTSPS with the assumtion that small cycles are prefered in the cycle cover, on the graph G_n^1, has weight larger than $\frac{1}{2}n \log_2 n$. Since the optimum TSP tour has length n, this gives an approximation in $\Omega(\log n)$.

The final construction

In this section we construct the family of graphs that gives the worst case preformance of the algorithm DTSPS. The graphs in the previous section have two main disadvantages: That they are symmetric and hence can be approximated by the algorithm due to Christofides and that the minimal cycle cover is not unique. The graphs defined in this section do not have these disadvantages.

Figure 2.3(b) shows a graph defined as follows:

Definition 11 *The distance function, $d_n(v_i, v_j)$, is induced by a graph G_n^D with $n = 2^m$ nodes arranged in a circle. Edges (v_{i-1}, v_i) in G_n^D have weight $w(v_{i-1}, v_i) = 1 + z_m(i)\epsilon$ where $0 < \epsilon < 1/n$. Edges (v_i, v_{i-2^k}) in G_n^D with $z_m(i) = k < m - 1$ have weight $w(v_i, v_{i-2^k}) = 2^k + (2^k - 1)\epsilon$.*

Definition 12 *Let $\mathcal{H}$ be a family of complete, asymmetric graphs, G_n, where G_n has $n = 2^m$ nodes and distance function $d_n(v_i, v_j)$.*

The next lemma shows that the graph G_n^D, inducing the distance function, obeys the triangle inequality. Hence the distance between v_i and v_j in G_n is equivalent to the edge weight of (v_i, v_j) in G_n^D whenever such an edge weight is specified in Definition 11. In fact, it also holds that $d(v_i, v_{i-2^k}) = 2^k + (2^k - 1)\epsilon$ even when $z_m(i) = k = m - 1$. We use this in several proofs below.

Lemma 3 If there is an edge (v_i, v_j) in the graph G_n^D then the shortest path in G_n^D from v_i to v_j is $d_n(v_i, v_j)$.

Proof 3 *Look at the edges $(v_i, v_{i\pm1})$. The largest edge has weight < 2, $d(v_{n-1}, v_0) = 1 + z_m(0)\epsilon = 1 + (m-1)\frac{1}{m} < 2$. A path over at least two edges has length larger than two (since the shortest edge has weight one). Hence the edge is always the shortest path.*

An edge (v_i, v_{i-2^k}), $z_m(i) = k$ has length $2^k + (2^k - 1)\epsilon$. There are two ways to get from v_i to v_{i-2^k} in G_n^D. One is clockwise around the circle and the path is $2^m + (2^m - 2)\epsilon - (2^k + (2^k - 1)\epsilon) \geq 2^k + (2^k - 1)\epsilon$ since $k < m$. The other is to use another edge $(v_i, v_{i-2^{k'}})$ with $k' > k$. But that edge has length larger than $2^k + (2^k - 1)\epsilon$ and thus the shortest path from v_i to v_{i-2^k} is the edge (v_i, v_{i-2^k}).

Lemma 4 *In a graph $G_n \in \mathcal{H}$, the maximum ratio, α, of edges in different directions is greater than $n/2$ if $\epsilon < \log_2 n$ and $n \geq 4$.*

Proof 4 *The edge (v_{n-1}, v_0) has by Definition 11 and Lemma 3 length $1+\epsilon(m-1)$. The edge (v_0, v_{n-1}) has length $2^m - 1 + (2^m - (m))\epsilon$ since the shortest path in G_n^D is clockwise around the circle. Thus the ratio is*

$$\frac{2^m - 1 + \epsilon(2^m - m)}{1 + \epsilon(m-1)} > \frac{n - 1 + 2^m/m - 1}{2} = \frac{n - 2 + 2^m/m}{2} \geq \frac{n}{2}$$

if $m \geq 2$ and $\epsilon < 1/\log_2 n$

The graph is clearly asymmetric and by Lemma 4 the maximum ratio between edges in different directions is linear in n. Frieze et al. show in their analysis of their data dependent algorithm that the approximation is in $O(\alpha)$. Hence, a graph G_n at least is not proven to be easily approximated by the data dependent algorithm.

Lemma 5 *In a graph $G_n \in \mathcal{H}$ the optimum TSP tour has weight $n + (n-2)\epsilon$.*

Proof 5 *The minimum TSP tour, $opt(G_n)$, is to traverse the nodes in clockwise order. Every edge has weight at least one which gives a weight of n. The "extra" weight is*

$$\sum_{i=1}^{\log_2 n} (\frac{n\epsilon}{2^i}) - \epsilon = n\epsilon(\sum_{i=1}^{\log_2 n} (\frac{1}{2^i})) - \epsilon = (n-2)\epsilon$$

and the total weight is $n + (n-2)\epsilon$.

Is the tour minimal? There are n edges in G_n of weight one. Only half of them can be in a TSP tour since they have opposite direction. There are $n/2$ edges of weight less than two, all induced edges have length greater than 2. The TSP tour consists of the n shortest edges possible in a TSP tour and is hence minimal.

For the following two lemmas we need a simpler distance function (Figure 2.3):

Definition 13 *The distance function $d_n^S(v_i, v_j)$ is induced by a symmetric graph G_n^{DS} with $n = 2^m$ nodes arranged in a circle. Adjacent nodes are connected by an edge. The weight of an edge is $w(v_{i-1}, v_i) = 1 + z_m(i)\epsilon$ where $0 < \epsilon < 1/n$. Let G_n^S be a complete, directed graph with $n = 2^m$ nodes and the distance function $d_n^S(v_i, v_j)$.*

The distance function in G_n^S is a "symmetrised" version of the distance function in G_n. The distance of (v_i, v_{i-1}) in G_n^{DS} is equal to the minimum of the weights of (v_i, v_{i-1}) and (v_{i-1}, v_i) in G_n^D (non-existent edges in G_n^D are here given weight ∞). Edges (v_i, v_{i-2^k}) in G_n^S with $z_m(i) = k < m-1$ are assigned to the induced distance $d(v_i, v_{i-2^k}) = 2^k + (2^k - 1)\epsilon$ which is the minimum of the weights of (v_i, v_{i-2^k}) and (v_{i-2^k}, v_i) in G_n^D. Hence edges in G_n^S are equal to or shorter than edges in G_n. If a cycle cover is minimal in G_n^S and all edges in the cycle cover have the same distance in G_n^S and in G_n then the cycle cover is minimal in G_n as well.

Lemma 6 *For $j < m$ and $n = 2^m$ it holds in G_n^S that $d(v_{i-2^j}, v_i) = 2^j + \epsilon \cdot (2^j - 1 + z_m(i \gg j))$. Here $\gg$ denotes a bitwise shift to the right padded with zeros on the left, i.e., $i \gg j = \lfloor i \cdot 2^{-j} \rfloor$*

Proof 6 *Consider the edge (v_{i-2^j}, v_i) in G_n^S. The path of edges $[i - 2^j, i]$ in G_n^{DS} has weight*

$$d(v_{i-2^j}, v_i) = \sum_{t=i-2^j+1}^{i} (1 + z_m(t)\epsilon) = 2^j + \epsilon \cdot \sum_{t=i-2^j+1}^{i} z_m(t)$$

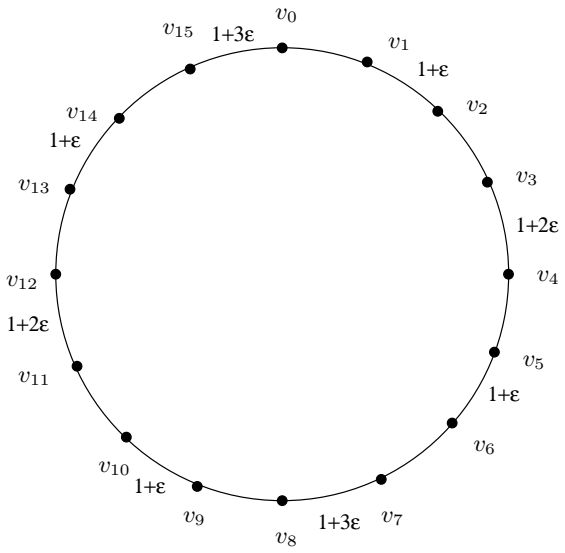

Figure 2.4: The graph G_{16}^{DS} inducing the distance function $d_n^S(v_i, v_j)$. For simplicity weights equal to one are omitted.

When the j least significant bits in t are zero $z_m(t) = z_m(t \gg j)+j$. The remaining terms sum up to

$$\sum_{t=1}^{2^j-1} z_m(t) = 2^j - 1 - j$$

If $j = m-1$ the path $[i, i-2^j]$ has equal weight to the path $[i-2^j, i]$. If $j < m-1$ the path $[i, i-2^j]$ has weight larger than 2^{m-1}. Since $\epsilon < 1/n$ this is larger than the weight of the path $[i-2^j, i]$. Hence the path $[i-2^j, i]$ is minimal and therefore induces the distance between v_{i-2^j} and v_i in G_n^S.

An edge (v_{i-2^j}, v_i) with fixed value of j has minimum distance if it is in the W-cactus since edges in the W-cactus has $z_m(i) = j$ which gives $z_m(i \gg j) = 0$ in the lemma above.

Lemma 7 *In the graph $G_n \in \mathcal{H}$ the algorithm DTSPS produces a W-cactus as spanning cactus and it has weight at least $n \log_2 n$.*

Proof 7 *First we show by induction that a minimum cycle cover in G_n^S is a W-cactus and has the desired weight. Then we show that the same cover exists in G_n. In the beginning $G_{n,0}^S$ consists of all n nodes v_i and $z_m(i) \geq 0$. Every other edge has length one and every other edge has at least one extra ϵ-distance added and the 2-cover is the unique minimal cycle cover. The edged in the cycles are*

$(v_i, v_{i-2^0}), (v_{i-2^0}, v_i)$ and the index i is odd or $z_m(i) = 0$. If the first node in every cycle is put in the subgraph, $G^S_{n,1}$ consists of nodes v_i with even index $i : z_m(i) \geq 1$. Suppose $G^S_{n,j}$ consists of all nodes v_i with $z_m(i) \geq j$ and that the cycle covers in $G^S_{n,r}$ for $r < j$ form a subgraph of the W-cactus. Every other edge has by Lemma 6 distance $2^j + (2^j - 1)\epsilon$ and every other has by at least one extra ϵ-distance added. The 2-cover is minimal. Select the first node in every cycle to be in $G^S_{n,j+1}$. Then the cycle cover in $G^S_{n,j}$ is a subgraph of the W-cactus and nodes v_i in $G^S_{n,j+1}$ have $z_m(i) \geq j+1$. By induction the 2-cover is minimal for every subgraph and it forms a W-cactus.

If there were no ϵ-weights the cactus would have weight $n \log_2 n$. Since all edges in the minimum cycle cover have the same weight in G_n the W-cactus is a minimum cycle cover in G_n as well.

Now we have a W-cactus as spanning cactus. The procedure SHORTCUT makes a TSP tour from the cactus.

Lemma 8 *In the graph G_n in the family $\mathcal{H}$ the approximation algorithm DTSPS gives a TSP tour of weight greater than $(n \log_2 n)/2$.*

Proof 8 *For a graph G_n the procedure DTSPS gives by Lemma 7 a W-cactus with weight greater than $n \log n$ as spanning cactus. The procedure SHORTCUT does not depend on the distance function. Hence if the spanning cactus is a W-cactus SHORTCUT will always return the same TSP tour. We show that the TSP tour in G^S_n with a W-cactus as spanning cactus has the desired weight and since every edge in G_n has at least the same weight as in G^S_n the TSP tour in G_n must have at least the same weight.*

From the proof of Lemma 7 the algorithm DTSPS given G^S_n, produces a W-cactus of weight greater than $n \log_2 n$. To show that the TSP tour produced by the algorithm has weight larger than half the weight of the W-cactus, we construct an injective function from the cycles in the cactus to the edges in the TSP tour such that each edge in the TSP tour has higher or equal distance than the longest edge in the corresponding cycle.

A cycle in the W-cactus (v_i, v_j) is mapped to an edge (v_t, v_j) in the TSP tour such that either $t = i$ or $t = j + (j - i) + 1$. The value of t is determined by the order in which edges are added to the TSP tour in SHORTCUT. Suppose DEPTH-FIRST is called with some even node s. The first cycle $(s, v) \in K$ processed in the loop is mapped to the edge (s, v). For the remaining iterations in the loop, the cycle (s, v) is mapped to the edge (t, v) where t was obtained from the call to DEPTH-FIRST in the previus iteration of the loop.

The mapping is obviously injective since DEPTH-FIRST visits every node exactly once. The first cycle is mapped to one edge in the cycle. If there are several cycles $(s, v) \in K$ in DEPTH-FIRST, s is even. Consider the cycle $(v_i, v_j) = (v_i, v_{i+2^k})$ which is not the first cycle chosen. The node $v = v_{i+2^{k+1}}$ was sent to DEPTH-FIRST in the previous recursion. Since $i + 2^{k+1}$ is even $t \leftarrow i + 2^{k+1} + 1$

was returned by Lemma 2. Hence, the cycle is mapped to the edge $(v_t, v_j) = (v_{i+2^{k+1}+1}, v_{i+2^k})$. *The difference between the indices of the nodes in this edge is* $\min\{|t-j|, n-|t-j|\} = 2^k+1$ *which is greater than the corresponding differnce for the cycle* (v_i, v_j) *since* $\min\{|i-j|, n-|i-j|\} = 2^k$. *Therefore Lemma 6 implies that* $d(v_t, v_j) \geq d(v_i, v_j)$.

Thus for every cycle there is an associated edge with length at least as high as the edges in the cycle and the TSP tour has weight at least half of the W-cactus in G_n^S.

By combining Lemma 8 and Lemma 5 we have proved Theorem 1.

2.4 Random Assumption on the Algorithm

Specification of the Approximation Algorithm

In order to analyse the algorithm we need to specify the arbitrary choices in the description by Frieze et al.:

1. An arbitrary node from every cycle in a cycle cover is chosen to be in the next subgraph (row 11 in ATSP). Choose the node randomly with equal probability for each node in the cycle.

2. The shortcuts made to transform the spanning cactus to a TSP tour in the procedure TOUR are in arbitrary order. Make the shortcuts by a depth-first search starting from the root of the cactus.

Constructing the graph

In this section we construct the family of symmetric graphs which has a worst case performance of the cycle covers produced by the algorithm ATSP.

Definition 14 *The distance function,* $d_n(v_i, v_j)$, *is induced by an undirected graph with* $n = 2m$ *nodes arranged in a circle. Adjacent nodes are connected and there are no other edges in the graph. The weight of an edge* $w(v_i, v_{i+1}) = 1 + 2^i\epsilon$ *with* $0 \leq i < m$ *and* $\epsilon < 1/2^{2n}$ *and* $w(v_i, v_{i+1}) = \delta$ *with* $m \leq i < n$ *and* $\delta < \epsilon/8m$.

Definition 15 *Let* $\mathcal{G}$ *be a family of complete, directed graphs* G_n *where* G_n *has* $n = 2m$ *nodes and the distance function* $d_n(v_i, v_j)$.

Notations. An edge in G_n with weight at least one is called a *large* edge and a node in the interval $[v_1, v_{m-1}]$ is called a *large* node. An edge shorter than one is called a *small* edge and a node in $[v_m, v_n]$ is called a *small* node (Figure 2.5). A small edge is connected to a small node.

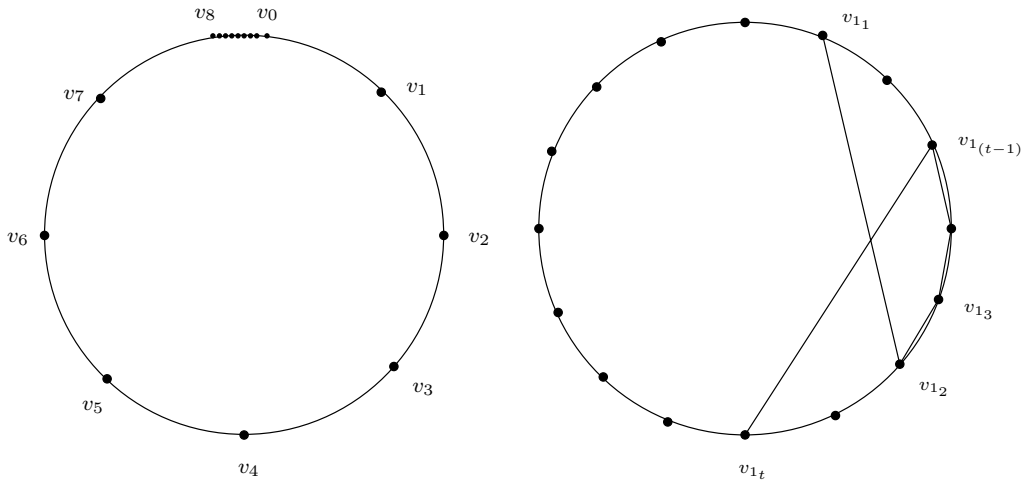

Figure 2.5: **Left (a)**: The graph inducing the distance function in G_{16}. (The figure does not have correct scale.) **Right (b)**: A cycle in a minimum cycle cover has no "twist".

Properties of the graphs

A more general family of graphs has distance function induced by nodes on a circle but with any positive weight on the edges. In this section we prove some properties for this more general family of graphs.

Definition 16 *$\mathcal{D}_n^G$ is the family of distance functions induced by an undirected graph with n nodes arranged in a circle. Adjacent nodes are connected by edges with positive weight, $w(v_i, v_{i+1}) > 0$, and there are no other edges in the graph. The distances are the shortest path in the graph and clearly obey the triangle inequality.*

Assumption 1 *Each edge on the circle obeys $w(v_i, v_{i+1}) < \frac{1}{2}\sum_{j=0}^{n-1} w(v_j, v_{j+1})$.*

Definition 17 *$\mathcal{G}_n^G$ is the family of complete, directed graphs with n nodes and a distance function in $\mathcal{D}_n^G$ which obeys Assumption 1.*

By the definition a graph in $\mathcal{G}_n^G$ obeys the triangle inequality.

Lemma 9 *The graph G_n is in $\mathcal{G}_n^G$ if $n \geq 6$.*

Proof 9 *The distance function in G_n is produced by nodes in a circle. A* large *edge has weight $w(v_i, v_{i+1}) < 1 + 2^{n/2}/2^{2n} < 1 + 1/2^n < 1 + 1/16$, if $n \geq 4$. If $n \geq 6$ then $1/2\sum_{j=0}^{n-1} w(v_j, v_{j+1}) > 3/2$ and the assumption holds.*

Lemma 10 *The minimum TSP tour in a graph* $G \in \mathcal{G}_n^G$ *has weight* $opt(G) = \sum_{i=0}^{n-1} w(v_i, v_{i+1})$.

Proof 10 *Any TSP tour in* G *can be mapped to a (not necessarily simple) tour on the circle by replacing every edge with a path on the circle. Let* T *be the tour on the circle corresponding to the minimum TSP tour. If* T *passes every edge at least once, then passing every edge exactly once gives minimum weight and* $opt(G) = \sum_{i=0}^{n-1} w(v_i, v_{i+1})$.

If T *does not pass every edge, suppose that* T *does not pass the edge* (v_i, v_{i+1})*, then all other edges must be passed to connect all nodes in* G*. Furthermore all edges have to be passed at least twice to make* T *a tour. By the assumption* $w(v_i, v_{i+1}) < 1/2 \sum_{j=0}^{n-1} w(v_j, v_{j+1})$ *and hence the weight of* T *is*

$$> 2(\sum_{j=0}^{n-1} w(v_j, v_{j+1}) - \frac{1}{2} \sum_{j=0}^{n-1} w(v_j, v_{j+1})) = \sum_{j=0}^{n-1} w(v_j, v_{j+1})$$

But then T *is not minimal which is a contradiction. Hence* T *passes every edge on the circle exactly once.*

Lemma 11 *The minimum TSP tour in* $G_n \in \mathcal{G}$ *with* $n = 2m \geq 6$ *has weight* $opt(G_n) < n/2 + 1/16$.

Proof 11 *By Lemma 9 and 10* $opt(G_n) = \sum_{i=0}^{n-1} w(v_i, v_{i+1}) = \sum_{i=0}^{m-1}(1 + \epsilon 2^i) + \sum_{i=m}^{n-1} \delta = m + (2^m - 1)\epsilon + m\delta < m + (2^{n/2} - 1)/2^{2n} + m/(8m2^{2n}) < n/2 + 1/16$. *Since* $\epsilon < 1/2^{2n}$ *and* $\delta < \epsilon/8m$.

Lemma 12 *Two nodes* v_i *and* v_j *in a graph* $G \in \mathcal{G}_n^G$ *obey* $d(v_i, v_j) = v_j - v_i$ *if* $v_j - v_i \leq opt(G)/2$.

Proof 12 *There are two paths from* v_i *to* v_{i+1} *on the circle and* $d(v_i, v_j) = v_j - v_i$ *or* $d(v_i, v_j) = v_i - v_j$*. Since* $v_j - v_i \leq 1/2 \sum_{j=0}^{n-1} w(v_j, v_{j+1}) < v_i - v_j$ *then* $v_j - v_i$ *is the shortest path.*

There is no "twist" in a cycle in the minimum cycle cover (Figure 2.5).

Lemma 13 *The nodes in a cycle* $C = (v_{1_1}, v_{1_2}, ..., v_{1_k})$ *in a minimum cycle cover in a graph* $G \in \mathcal{G}_n^G$*, can always be written in clockwise order with the lowest index first.*

Proof 13 *Suppose that the nodes in* C *can not be written in clockwise order.* C *connects a subset of the nodes in* G*. Every cycle in* G *can be transformed to a (not necessarily simple) tour,* T*, on the circle.*

If all edges on the circle are in T *at least once, then the weight is minimum if they are in the tour exactly once and the nodes in* C *can be written in clockwise order.*

If some edge is not in T. T must be a connected component on the circle. All edges in T are in T at least twice to make it a tour. Minimum weight is obtained when all edges are in the tour exactly twice and then the nodes can be written in clockwise order.

Hereafter we will always assume that the nodes in a cycle are written in clockwise order with the lowest index first. A minimum cycle cover is either a TSP tour or has only short cycles.

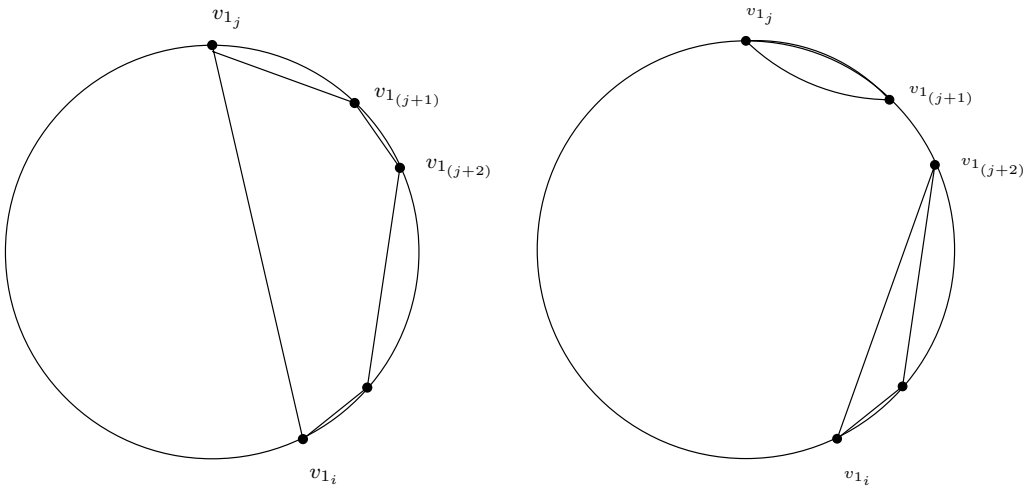

Figure 2.6: **Left (a)**: A cycle which is not in a minimum cycle cover. **Right (b)**: The same graph with reduced weight.

Lemma 14 *A minimum cycle cover in a graph $G \in \mathcal{G}_n^G$ with weight less than $opt(G)$ consists of cycles with 2 or 3 nodes.*

Proof 14 *Consider a cycle $C = (v_{1_1}, v_{1_2}, ..., v_{1_k})$ in the minimum cycle cover (with the nodes in clockwise order and with the lowest index first), with weight less than the TSP tour, $w(C) < opt(G)$, and with more than three nodes, $3 < k$ (Figure 2.6). Select the largest edge, (v_{1_i}, v_{1_j}). Since $w(C) < opt(G)$, the distance $d(v_{1_i}, v_{1_j})$ is not induced by $d[v_{1_i}, v_{1_j}]$. Hence it must be induced by the edges in opposite direction $d[v_{1_j}, v_{1_i}]$. Replace the edge (v_{1_i}, v_{1_j}) with $(v_{1_j}, v_{1_{(j+1)}})$ and $(v_{1_{(j+2)}}, v_{1_i})$ (Figure 2.6). Now C is divided into two cycles and the weight is reduced with $2w(v_{1_{(j+1)}}, v_{1_{(j+2)}})$. Hence a cycle can connect at most 3 nodes.*

The next lemma shows that the cycles in a minimum cycle cover do not overlap (Figure 2.7).

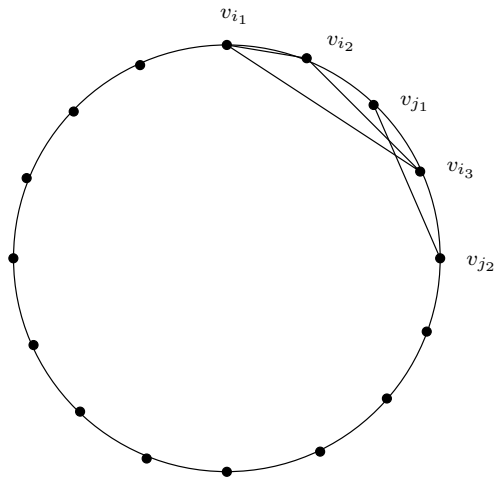

Figure 2.7: In a minimum cycle cover cycles are not "overlapping".

Lemma 15 *The nodes in two cycles* $C_1 = (v_{1_1}, v_{1_2}, ..., v_{1_k})$ *and* $C_2 = (v_{2_1}, v_{2_2}, ... , v_{2_m})$ *in a minimum cycle cover in a graph* $G \in \mathcal{G}_n^G$ *can be written such that* $v_{1_1}, v_{1_2}, ... , v_{1_s}, v_{2_1}, v_{2_2}, ... , v_{2_m}, v_{1_{(s+1)}}, ..., v_{1_k}$ *are in clockwise order.*

Proof 15 *A cycle in* G *can be transformed to a tour on the circle. Let* T_1 *and* T_2 *be the tours corresponding to* C_1 *and* C_2*. Both cycles have weight less than* $\sum_{j=0}^{n-1} w(v_j, v_{j+1})$*, otherwise one large cycle is minimal. Hence both tours are passing the edges on a segment of the circle exactly twice. If the segments are overlapping the weight of the cycles are not minimal. Hence in a minimum cycle cover the cycles are not overlapping and then they can be written on the desired form.*

Later we will need the following lemma in a later proof.

Lemma 16 *Any partitioning of the edges defining the distance function in* $G_n \in \mathcal{G}$ *in two sets,* E_1 *and* E_2*, obey* $|\sum_{e_i \in E_1} w(e_i) - \sum_{e_i \in E_2} w(e_i)| \geq \epsilon/2$

Proof 16 *There are* m large *edges with weight larger than one. The total sum of* ϵ*-weights is* $(2^m - 1)\epsilon < (2^m - 1)/2^{4m} < 1/2^{3m}$*. The total sum of* δ*-weights is* $m\delta = m\epsilon/8m = \epsilon/8 < 1/2^{3+4m}$*. We will try to construct two sets* E_1 *and* E_2 *with equal weight. Since the sum of* ϵ*- and* δ*-weights is less than* $1/2^n$ *the large edges must be equally distrubuted between the sets. The number of* ϵ*-weights in a set is an* m*-bit binary number where a bit* i *is one if the edge* v_i *is in the set. The binary numbers for the sets are bitwise complementary and can never be equal. The smallest difference is one. If we put all* δ*-edges in the smaller set the difference is still* $\epsilon(1 - 1/8) > 7\epsilon/8$*.*

To make the notation clear we define:

Definition 18 *For the algorithm ATSP and the graph $G_n = G_{n,0}$, the subgraph remaining after the first cycle cover is found is denoted by $G_{n,1}$ and the subgraph remaining after the i:th iteration is denoted by $G_{n,i}$.*

Lemma 17 *If $i < \log_3 m$ then the minimum cycle cover in the graph $G_{2m,i}$ consists of cycles with two or three nodes.*

Proof 17 *If the minimum cycle cover has weight $< opt(G)$ by Lemma 14 it consists of short cycles. Hence a cycle cover consisting of one cycle must have weight $opt(G)$.*

Assume that all cycles in a minimum cycle cover in $G_{2m,j}$ with $j < i$ connect at most three nodes.

If the number of nodes is even then divide the edges between the nodes in two sets; E_1 is every second edge and E_2 is the other edges. The sets is a partitioning of the edges defining the distance function and by Lemma 16 the weight difference is at least $\epsilon/2$. Take the set with smaller weight, which is at most $(opt(G_n) - \epsilon/2)/2$. Compleeting these 2-cycles gives a cover with weight less than $opt(G_n)$, and hence the TSP-tour is not minimal.

If the number of nodes is odd then since $i < \log_3 m$ and the assumption that all previous cycles connect at most three nodes there are at least two small nodes. Remove one small node and there is an even number of remaining nodes. By the argument above there is a 2-cycle cover connecting the nodes with weight less than $opt(G_n) - \epsilon/2$. The largest possible distance between two small nodes is $\delta \cdot m = \epsilon m/8m = \epsilon/8$. Connecting the removed node to the cycle containingits small neighbour increases the weight of the cycle cover with at most $\epsilon/4$. Hence the cycle cover with small cycles has weight less than the TSP tour and the TSP tour is not minimal.

In a minimum cycle cover with small cycles by Lemma 14 a cycle connects at most three nodes. Hence the assumption that cycles in $G_{2m,j}$ are short holds. After $\log_3 m$ iterations there might be only one small node in the graph and the TSP-tour might be minimal.

The minimum cycle covers

In this section we will show that the weight of the spanning cactus produced by ATSP on G_n is heavy. First we show that when two or three cacti are connected the weight grows with essentially the length of the shortest cactus. Then we show how fast the weight grows with the length of the cactus. To do this we need a more simple graph, G_n^S.

By Lemma 15 cycles in a minimum cycle cover in a graph $G \in \mathcal{G}_n^G$ never overlap. Hence a cactus formed by the algorithm ATSP connects all nodes in an interval $[v_a, v_b]$.

Definition 19 *After j iteration of ATSP on a graph $G \in \mathcal{G}_n^G$ a cactus K_i is produced. K_i connects all nodes on the interval $[v_a, v_b]$. The* length *of the cactus is the length of the interval, $l(K_j) = d_n[v_a, v_b]$.*

When several cacti are connected we need a definition for which side they are nearest neighbours on. We use the interval of connected nodes for each cactus for the definition and use the knowledge from Lemma 14 that the cycles are short. The definition is not well defined if all nodes in the graph are connected.

Definition 20 *A cactus $K_{i,j}$ is produced by j iterations of the algorithm ATSP on a graph $\mathcal{G}_n^G$.*

If $K_{i,j}$ connects two subcacti, $K_{k,j-1}$ and $K_{l,j-1}$ and $K_{k,j-1}$ connects all node in the interval $[v_a, v_b]$ and $K_{l,j-1}$ all node in the interval $[v_c, v_d]$ then the subcacti are connected in the given order *if $K_{i,j}$ connects all nodes in $[v_a, v_d]$.*

If $K_{i,j}$ connects three subcacti $K_{k,j-1}$, $K_{l,j-1}$ and $K_{m,j-1}$ and $K_{k,j-1}$ connects all nodes in $[v_a, v_b]$, $K_{l,j-1}$ connects $[v_c, v_d]$ and $K_{m,j-1}$ connects $[v_e, v_f]$ then the subcacti are connected in the given order *if $K_{i,j}$ connects all nodes in the interval $[v_a, v_f]$.*

To simplify some proofs we need a simple distance function.

Definition 21 *The distance function, $d_n^S(v_i, v_j)$, is induced by an undirected graph with $n = 2m$ nodes arranged in a circle. Adjacent nodes are connected and there are no other edges in the graph. The weight of an edge $w(v_i, v_{i+1}) = 1$ with $0 \leq i < m$ and $w(v_i, v_{i+1}) = \gamma$ with $m \leq i < n$ and $\gamma < \frac{1}{n2^{2n+3}}$.*

Definition 22 *Let G_n^S be a complete, directed graph with $n = 2m$ nodes and the distance function $d_n^S(v_i, v_j)$.*

The simple distance function $d_n^S(v_i, v_j)$ is similar to $d_n(v_i, v_j)$ with the ϵ weights removed. It is simple to see that the weight of a cactus K^S in G_n^S is less than the weight of the same cactus K in G_n, $w(K) \geq w(K^S)$. Also for this distance function $G_n^S \in \mathcal{G}_n^G$ if $n \geq 6$.

Definition 23 *For a cactus, K, which connects nodes in $[v_a, v_b]$ in a graph $G \in \mathcal{G}_n^G$ and v is the node in the next iteration, $\delta(K) = min(d(v, v_a), d(v, v_b))$.*

We would like the weight, $w(K)$, of a cactus to grow with a constant fraction of the length, $l(K)$, in every iteration of ATSP. Unfortunately this is not the case for example when the nodes in the next iteration are close to each other. By adding the term $\delta(K)$ we get the desired behaviour.

Lemma 18 *A cactus $K_{i,j+1}$ is produced by $j+1$ iterations of the algorithm ATSP in a graph G_n^S. If $K_{i,j+1}$ only connects large nodes, only contains cycles with 2 or*

3 nodes and $l(K_{i,j+1}) < opt(G_n^S)/2$ *then it either connects two subcacti* $K_{i,j}$ *and* $K_{m,j}$ *or three subcacti* $K_{i,j}$, $K_{k,j}$ *and* $K_{m,j}$ *in the given order and*

$$\begin{aligned} w(K_{i,j+1}) + \delta(K_{i,j+1}) \geq \\ w(K_{i,j}) + \delta(K_{i,j}) + w(K_{m,j}) + \delta(K_{m,j}) + min(l(K_{i,j}), l(K_{m,j})) + 2 \end{aligned} \tag{2.1}$$

or

$$\begin{aligned} w(K_{i,j+1}) + \delta(K_{i,j+1}) \geq w(K_{i,j}) + \delta(K_{i,j}) + w(K_{k,j}) + \delta(K_{k,j}) + \\ w(K_{m,j}) + \delta(K_{m,j}) + 1.5l(K_{k,j}) + min(l(K_{i,j}), l(K_{m,j})) + 4 \end{aligned} \tag{2.2}$$

Proof 18 *Since* $l(K_{i,j+1}) < opt(\mathcal{G}_n^G)/2$ *by Lemma 12 if* $p > q$ *then* $d(v_p, v_q) = d[v_p, v_q]$.

For a cactus $K_{i,j+1}$ *there are two cases: the first case is when it connects two subcacti* $K_{i,j}$ *and* $K_{m,j}$ *in the given order. Assume that* $l(K_{i,j}) \geq l(K_{m,j})$. *The cactus* $K_{i,j+1}$ *has weight*

$$w(K_{i,j+1}) = w(K_{i,j}) + w(K_{m,j}) + 2d(v_{i,j}, v_{m,j})$$

and

$$w(K_{n,j+1}) + \delta(K_{i,j+1}) = w(K_{i,j}) + w(K_{m,j}) + 2d(v_{m,j}, v_{i,j}) + \delta(K_{i,j+1})$$

Assume that the cactus K *connects nodes in the interval* $[v_a, v_b]$ *and* v *is the node in the next iteration. Hereafter we denote the length of the interval* $l_a(K) = d[v_a, v]$ *and* $l_b(K) = d[v, v_b]$.

$$d(v_{i,j}, v_{m,j}) = l_b(K_{i,j}) + l_a(K_{m,j}) + 1$$

If the node in the next iteration $v_{i,j+1} = v_{m,j}$ *then the distance* $\delta(K_{i,j+1}) = l_b(K_{m,j})$ *since* $l(K_{i,j}) \geq l(K_{m,j})$.

$$\begin{aligned} 2d(v_{i,j}, v_{m,j}) + \delta(K_{i,j+1}) \geq \\ 2(l_b(K_{i,j}) + l_a(K_{m,j}) + 1) + l_b(K_{m,j}) = \\ l(K_{m,j}) + 2l_b(K_{i,j}) + l_a(K_{m,j}) + 2 \geq \\ l(K_{m,j}) + 2\delta(K_{i,j}) + \delta(K_{m,j}) + 2 \end{aligned}$$

If the node in the next iteration $v_{i,j+1} = v_{i,j}$ *and if the distance* $\delta(K_{i,j+1}) = l_a(K_{i,j})$ *then*

$$\begin{aligned} 2d(v_{i,j}, v_{m,j}) + \delta(K_{i,j+1}) \geq \\ 2(l_b(K_{i,j}) + l_a(K_{m,j}) + 1) + l_a(K_{i,j}) = \\ l(K_{i,j}) + l_b(K_{i,j}) + 2l_a(K_{m,j}) + 2 \geq \\ l(K_{i,j}) + \delta(K_{i,j}) + 2\delta(K_{m,j}) + 2 \end{aligned}$$

If the distance $\delta(K_{i,j+1}) = l_b(K_{i,j}) + 1 + l(K_{m,j})$ *then*

$$\begin{aligned} 2d(v_{i,j}, v_{m,j}) + \delta(K_{i,j+1}) \geq \\ 2(l_b(K_{i,j}) + l_a(K_{m,j}) + 1) + l_b(K_{i,j}) + 1 + l(K_{m,j}) = \\ l(K_{m,j}) + 3l_b(K_{i,j}) + 2l_a(K_{m,j}) + 3 \geq \\ l(K_{m,j}) + 3\delta(K_{i,j}) + 2\delta(K_{m,j}) + 3 \end{aligned}$$

Hence Inequality 2.1 holds.

The second case is when $K_{i,j+1}$ *connects three subcacti,* $K_{i,j}$, $K_{k,j}$ *and* $K_{m,j}$, *in the given order. Suppose* $l(K_{m,j}) \leq l(K_{i,j})$. *The cactus* $K_{i,j+1}$ *has weight:*

$$w(K_{i,j+1}) = w(K_{i,j}) + w(K_{k,j}) + w(K_{m,j}) + 2d(v_{i,j}, v_{m,j})$$

and

$$w(K_{i,j+1}) + \delta(K_{i,j+1}) = w(K_{i,j}) + w(K_{k,j}) + w(K_{m,j}) + 2d(v_{i,j}, v_{m,j}) + \delta(K_{i,j+1})$$

The distance between the nodes in the next iterations is

$$d(v_{i,j}, v_{m,j}) = l_b(K_{i,j}) + l(K_{k,j}) + l_a(K_{m,j}) + 2$$

If the node in the next iteration $v_{i,j+1} = v_{m,j}$ *then* $\delta(K_{i,j+1}) = l_b(K_{m,j})$ *and*

$$\begin{aligned} 2d(v_{i,j}, v_{m,j}) + \delta(K_{i,j+1}) \geq \\ 2(l_b(K_{i,j}) + l(K_{k,j}) + l_a(K_{m,j}) + 2) + l_b(K_{m,j}) = \\ 2l(K_{k,j}) + l(K_{m,j}) + 2l_b(K_{i,j}) + l_a(K_{m,j}) + 4 \geq \\ 2l(K_{k,j}) + l(K_{m,j}) + 2\delta(K_{i,j}) + \delta(K_{m,j}) + 4 \end{aligned}$$

If $v_{i,j+1} = v_{i,j}$ *and if* $\delta(K_{i,j+1}) = l_a(K_{i,j})$ *then*

$$\begin{aligned} 2d(v_{i,j}, v_{m,j}) + \delta(K_{i,j+1}) \geq \\ 2(l_b(K_{i,j}) + l(K_{k,j}) + l_a(K_{m,j}) + 2) + l_a(K_{i,j}) = \\ 2l(K_{k,j}) + l(K_{i,j}) + l_b(K_{i,j}) + 2l_a(K_{m,j}) + 4 \geq \\ 2l(K_{k,j}) + l(K_{i,j}) + \delta(K_{i,j}) + 2\delta(K_{m,j}) + 4 \end{aligned}$$

Otherwise if $\delta(K_{i,j+1}) = l_b(K_{i,j}) + l(K_{k,j}) + l(K_{m,j}) + 2$ *then*

$$\begin{aligned} 2d(v_{i,j}, v_{m,j}) + \delta(K_{i,j+1}) \geq \\ 2(l_b(K_{i,j}) + l(K_{k,j}) + l_a(K_{m,j}) + 2) + l_b(K_{i,j}) + l(K_{k,j}) + l(K_{m,j}) + 2 = \\ 3l(K_{k,j}) + l(K_{m,j}) + 3l_b(K_{i,j}) + l_b(K_{m,j}) + 6 \geq \\ 3l(K_{k,j}) + l(K_{m,j}) + 3\delta(K_{i,j}) + \delta(K_{m,j}) + 6 \end{aligned}$$

If the node in the next iteration $v_{i,j+1} = v_{k,j}$ *then* $\delta(K_{i,j+1}) \geq \delta(K_{k,j}) + l(K_{m,j}) + 1$ *and*

$$\begin{aligned} 2d(v_{i,j}, v_{m,j}) + \delta(K_{i,j+1}) \geq \\ 2(l_b(K_{i,j}) + l(K_{k,j}) + l_a(K_{m,j}) + 2) + \delta(K_{k,j}) + l(K_{m,j}) + 1 = \\ 2l(K_{k,j}) + l(K_{m,j}) + 2\delta(K_{i,j}) + \delta(K_{k,j}) + 2\delta(K_{m,j}) + 5 \end{aligned}$$

Hence Inequality 2.2 holds.

Next we show how the weight of a cactus, $w(K)$, grows with the length, $l(K)$.

Lemma 19 *A cactus* K *is formed by* $j+1$ *iterations of ATSP in* G_n^S *and has length* $l(K)$*. If* K *only connects large nodes, if* $l(K) \leq opt(G_n^S)/2$ *and if it has cycles of length* 2 *and* 3 *it obeys* $w(K) + \delta(K) \geq l(K)(a \log_2 l(K) - bj)$*. Where* $a = \frac{1}{4}$ *and* $b = \frac{24}{100}$*.*

Proof 19 *The proof is by induction. Since* $l(K) \leq opt(G_n^S)/2$ *by Lemma 12 if* $p > q$ *then* $d(v_p, v_q) = d[v_p, v_q]$ *for all nodes* v_q *and* $v_p \in K_{i,j+1}$*. In the first iteration* $j = 0$ *and single nodes are connected. There are two possible cases: first a 2-cycle. By Lemma 15 a cycle in a minimum cycle cover connects nearest neighbours and* $K = (v_i, v_{i+1})$*,* $w(K) = 2$*,* $l(K) = 1$ *and* $\delta(K) = 0$*. Then*

$$\begin{aligned} w(K) + \delta(K) = 2 \\ l(K)(a \log_2 l(K) - bj) = l(a \log_2 1 - b \cdot 0) = 0 < 2 \end{aligned}$$

The second case is when three nodes are connected. $K = (v_i, v_{i+1}, v_{i+2})$*,* $w(K) = 4$*,* $l(K) = 2$ *and* $\delta(K) \leq 1$*. Then*

$$\begin{aligned} w(K) + \delta(K) \geq 4 \\ l(K)(a \log_2 l(K) - bj) = 2(a \log_2 2 - b \cdot 0) = 2a = 1/2 < 4 \end{aligned}$$

Hence the lemma is true for $j = 0$*.*

Assume that the relation is true for all values up to some j*. There are two inductive steps. The first is when* $K_{i,j+1}$ *connects two subcacti* $K_{i,j}$ *and* $K_{m,j}$ *in that order. Assume that* $l(K_{i,j}) \geq l(K_{m,j})$*. Since an edge between two large nodes in* G_n^S *has length* 1*,* $l(K_{i,j+1}) = l(K_{i,j}) + 1 + l(K_{m,j})$*. By Lemma 18 and by the induction hypothesis*

$$\begin{aligned} w(K_{i,j+1}) + \delta(K_{i,j+1}) = \\ w(K_{i,j}) + \delta(K_{i,j}) + w(K_{m,j}) + \delta(K_{m,j}) + min(l(K_{i,j}), l(K_{m,j})) + 2 \geq \\ l(K_{i,j+1})(a \log_2 l(K_{i,j}) - bj) + l(K_{m,j})(a \log_2 l(K_{m,j}) - bj) + l(K_{m,j}) + 2 \end{aligned} \quad (2.3)$$

Our aim is to show that this is greater than:

$$(l(K_{i,j}) + l(K_{m,j}) + 1)(a \log_2(l(K_{i,j}) + l(K_{m,j}) + 1) - b(j+1))$$

Expanding Sum 2.3 gives:

$$\begin{aligned}(l(K_{i,j})+l(K_{m,j})+1)(a\log_2(l(K_{i,j})+l(K_{m,j})+1)-b(j+1))-\\ l(K_{i,j})a(\log_2(l(K_{i,j})+l(K_{m,j})+1)-\log_2 l(K_{i,j}))-\\ l(K_{m,j})a(\log_2(l(K_{i,j})+l(K_{m,j})+1)-\log_2 l(K_{m,j}))-\\ a\log_2(l(K_{i,j})+l(K_{m,j})+1)+l(K_{i,j})b+l(K_{m,j})b+b(j+1)+l(K_{m,j})+2\end{aligned}$$

Hence we need to show that:

$$\begin{aligned}l(K_{i,j})a(\log_2(l(K_{i,j})+l(K_{m,j})+1)-\log_2 l(K_{i,j}))+\\ l(K_{m,j})a(\log_2(l(K_{i,j})+l(K_{m,j})+1)-\log_2 l(K_{m,j}))+\\ a\log_2(l(K_{i,j})+l(K_{m,j})+1)-\\ l(K_{i,j})b-l(K_{m,j})b-b(j+1)-l(K_{m,j})-2\leq 0\end{aligned}\tag{2.4}$$

Let $l(K_{m,j})t = l(K_{i,j})$ for $t \geq 1$. The first three terms in Equation 2.4 can be simplified to:

$$\begin{aligned}l(K_{i,j})a(\log_2(l(K_{i,j})+l(K_{m,j})+1)-\log_2 l(K_{i,j}))=\\ l(K_{i,j})a(\log_2(l(K_{i,j})+l(K_{i,j})/t+1)-\log_2 l(K_{i,j}))=\\ l(K_{i,j})a(\log_2(1+1/t+1/l(K_{i,j}))+\log_2 l(K_{i,j})-\log_2 l(K_{i,j}))=\\ l(K_{i,j})a(\log_2(1+1/t+1/l(K_{i,j}))\end{aligned}$$

$$\begin{aligned}l(K_{m,j})a(\log_2(l(K_{i,j})+l(K_{m,j})+1)-\log_2 l(K_{m,j}))=\\ l(K_{m,j})a(\log_2(l(K_{i,j})+l(K_{m,j})+1)/l(K_{m,j}))=\\ l(K_{m,j})a\log_2(t+1+1/l(K_{m,j})=\\ l(K_{i,j})/t\cdot a\log_2(t+1+1/l(K_{m,j}))\end{aligned}$$

$$a\log_2(l(K_{i,j})+l(K_{m,j})+1)\leq a\log_2(3l(K_{i,j}))=a\log_2(l(K_{i,j}))+a\log_2 3$$

Inserted in Inequality 2.4 and since $l(K_{m,j}) \geq 1$, $l(K_{i,j}) \geq t$, Inequality 2.4 is implied by

$$\begin{aligned}l(K_{i,j})a\log_2(1+1/t+1/l(K_{i,j}))+l(K_{i,j})/t\cdot a\log_2(t+1+1/l(K_{m,j}))+\\ a\log_2(l(K_{i,j}))+a\log_2 3-bl(K_{i,j})-bl(K_{m,j})-b(j+1)-l(K_{m,j})-2\leq\\ l(K_{i,j})(a\log_2(1+2/t)+1/t\cdot a\log_2(t+2)+\\ a\log_2(l(K_{i,j}))/l(K_{i,j})-b-b/t-1/t)+a\log_2 3-b(j+1)-2\end{aligned}$$

The last three terms, independent of the length of the cactus, are

$$a\log_2 3-b(j+1)-2<-1\leq 0$$

Hence it is enough to show that

$$a\log_2(1+2/t)+(1/t)\cdot a\log_2(t+2)+ \\ a\log_2(l(K_{i,j}))/l(K_{i,j})-b-b/t-1/t\leq 0 \tag{2.5}$$

Suppose $x\leq t\leq x'$ *then the left hand side of Inequality 2.5 is less than*

$$a\log_2(1+2/x)+(1/x)a\log_2(x+2)+a\log_2 e/e-b-b/x'-1/x' \tag{2.6}$$

For different values of the variables x *and* x' *the Equation 2.6 is bounded from above*

x	x'	*Expression 2.6 <*
1	*1.8*	*-0.003*
1.8	*2.8*	*-0.01*

Suppose that $e<x\leq t\leq x'$ *then since the function* $\log_2 t/t$ *is decreasing for* $t\geq e$ *the left hand side of Inequality 2.5 is less than*

$$a\log_2(1+2/x)+1/xa\log_2(x+2)+a\log_2 x'/x'-b-b/x'-1/x' \tag{2.7}$$

For different values of the variables x *and* x' *Expression 2.7 is bounded from above*

x	x'	*Expression 2.7 <*
2.8	*4*	*-0.02*
4	*6*	*-0.01*
6	*12*	*-0.006*
12	∞	*-0.05*

The second inductive step is when $K_{i,j+1}$ *connects three subcacti* $K_{i,j}$, $K_{k,j}$ *and* $K_{m,j}$ *in the given order. Suppose that* $l(K_{i,j})\geq l(K_{m,j})$. *By Lemma 18 and by the induction hypothesis:*

$$\begin{aligned} w(K_{i,j+1})+\delta(K_{i,j+1})\geq \\ w(K_{i,j})+\delta(K_{i,j})+w(K_{k,j})+\delta(K_{k,j})w(K_{m,j})+\delta(K_{m,j})+ \\ 1.5l(K_{k,j})+min(l(K_{i,j}),l(K_{m,j}))+4\geq \\ l(K_{i,j})(a\log_2 l(K_{i,j})-bj)+l(K_{k,j})(a\log_2 l(K_{k,j})-bj)+ \\ l(K_{m,j})(a\log_2 l(K_{m,j})-bj)+1.5l(K_{k,j})+l(K_{m,j})+4 \end{aligned} \tag{2.8}$$

We want to show that this is greater than:

$$(l(K_{i,j})+l(K_{k,j})+l(K_{m,j})+2)(a\log_2(l(K_{i,j})+l(K_{k,j})+l(K_{m,j})+2)-b(j+1))$$

Expanding Sum 2.8 gives:

$$\begin{aligned} (l(K_{i,j})+l(K_{k,j})+l(K_{m,j})+2)(a\log_2(l(K_{i,j})+l(K_{k,j})+l(K_{m,j})+2)-b(j+1))- \\ l(K_{i,j})a(\log_2(l(K_{i,j})+l(K_{k,j})+l(K_{m,j})+2)-\log_2 l(K_{i,j}))- \\ l(K_{k,j})a(\log_2(l(K_{i,j})+l(K_{k,j})+l(K_{m,j})+2)-\log_2 l(K_{k,j}))- \\ l(K_{m,j})a(\log_2(l(K_{i,j})+l(K_{k,j})+l(K_{m,j})+2)-\log_2 l(K_{m,j}))- \\ 2a(\log_2(l(K_{i,j})+l(K_{k,j})+l(K_{m,j})+2)+ \\ l(K_{i,j})b+l(K_{k,j})b+l(K_{m,j})b+2b(j+1)+1.5l(K_{k,j})+l(K_{m,j})+4 \end{aligned}$$

Our goal is to prove that:

$$\begin{aligned} l(K_{i,j})a(\log_2(l(K_{i,j})+l(K_{k,j})+l(K_{m,j})+2)+\log_2 l(K_{i,j}))+\\ l(K_{k,j})a(\log_2(l(K_{i,j})+l(K_{k,j})+l(K_{m,j})+2)+\log_2 l(K_{k,j}))+\\ l(K_{m,j})a(\log_2(l(K_{i,j})+l(K_{k,j})+l(K_{m,j})+2)-\log_2 l(K_{m,j}))+\\ 2a(\log_2(l(K_{i,j})+l(K_{k,j})+l(K_{m,j})+2)-\\ l(K_{i,j})b-l(K_{k,j})b-l(K_{m,j})b-2b(j+1)-1.5l(K_{k,j})-l(K_{m,j})-4\leq 0 \end{aligned} \quad (2.9)$$

To make the term $1.5l(K_{k,j})$ *as small as possible let* $l(K_{k,j}) \leq l(K_{m,j})$ *. Let* $t_m l(K_{m,j}) = l(K_{i,j})$ *and* $t_k l(K_{k,j}) = l(K_{m,j})$*. The first four terms in Inequality 2.9 can be simplified to:*

$$\begin{aligned} l(K_{i,j})a(\log_2(l(K_{i,j})+l(K_{k,j})+l(K_{m,j})+2)-\log_2 l(K_{i,j}))=\\ l(K_{i,j})a\log_2(1/t_k+t_m+1+2/l(K_{i,j})) \end{aligned}$$

$$\begin{aligned} l(K_{k,j})a(\log_2(l(K_{i,j})+l(K_{k,j})+l(K_{m,j})+2)-\log_2 l(K_{k,j}))=\\ l(K_{k,j})a\log_2(t_k+t_m t_k+1+2/l(K_{k,j})) \end{aligned}$$

$$\begin{aligned} l(K_{m,j})a(\log_2(l(K_{i,j})+l(K_{k,j})+l(K_{m,j})+2)-\log_2 l(K_{m,j}))=\\ l(K_{m,j})a\log_2(1/t_m+1/t_m t_k+1+2/l(K_{m,j}) \end{aligned}$$

$$\begin{aligned} a\log_2(l(K_{i,j})+l(K_{k,j})+l(K_{m,j})+2)=\\ a\log_2 l(K_{m,j})(1/t_m+1/t_k t_m+1+2)=\\ a\log_2 l(K_{m,j})+a\log_2 5 \end{aligned}$$

Inserted in Inequality 2.9:

$$\begin{aligned} l(K_{i,j})a\log_2(1/t_k+t_m+1+2/l(K_{i,j}))+\\ l(K_{k,j})a\log_2(t_k+t_m t_k+1+2/l(K_{k,j}))+\\ l(K_{m,j})a\log_2(1/t_m+1/t_m t_k+1+2/l(K_{m,j}))+\\ 2a\log_2 l(K_{m,j})+2a\log_2 5-l(K_{i,j})b-l(K_{k,j})b-\\ l(K_{m,j})b-2b(j+1)-1.5l(K_{k,j})-l(K_{m,j})-5\leq\\ l(K_{m,j})(1/t_m a\log_2(1/t_k+t_m+1+2/l(K_{i,j}))+\\ 1/(t_k t_m)a\log_2(t_k+t_m t_k+1+2/l(K_{k,j}))+\\ a\log_2(1/t_m+1/t_m t_k+1+2/l(K_{m,j}))+\\ 2a\log_2 l(K_{m,j})/l(K_{m,j})\\ -b(1/t_m+1/t_m t_k+1+2(j+1)/l(K_{m,j})))-\\ 1.5/(t_k t_m)-1/t_m)+2a\log_2 5-4 \end{aligned}$$

The last two terms, independent of the length of the cactus, are

$$2a\log_2 5 - 4 < -2 < 0$$

Hence it is enough to show that

$$\begin{aligned} 1/t_m a\log_2(1/t_k + t_m + 1 + 2/l(K_{i,j})) + \\ 1/(t_k t_m)a\log_2(t_k + t_m t_k + 1 + 2/l(K_{k,j})) + \\ a\log_2(1/t_m + 1/t_m t_k + 1 + 2/l(K_{m,j})) + \\ 2a\log_2 l(K_{m,j})/l(K_{m,j}) - b(1/t_m + 1/t_m t_k + 1 + 2(j+1)/l(K_{m,j}))) - \\ 1.5/(t_k t_m) - 1/t_m) \leq 0 \end{aligned} \quad (2.10)$$

Since $l(K_{k,j}) \geq 1$ *the length* $l(K_{m,j}) = t_k t_m l(K_{k,j}) \geq t_k t_m$. *The term* $(\log_2 x)/x \leq (\log_2 e)/e$ *for all* x *and for* $x \geq e$ *the function is decreasing. Hence for* $t_k t_m \geq e$, $(\log_2 l(K_{m,j})/l(K_{m,j}) \leq (\log_2 t_k t_m)/t_k t_m$. *Since* $l(K) \leq 3^j$, $j \geq \log_3 l(K) \geq \log_3 t_k t_m \geq 1$

Assume that $1 = x_m \leq t_m \leq x'_m$ *and* $1 = x_k \leq t_k < x'_k$. *Inequality 2.10 is less than or equal to*

$$\begin{aligned} 1/x_m(a\log_2(1/x_k + x_m + 1 + 2/x_k)) + \\ 1/(x_k x_m)(a\log_2(x_k + x_m x_k + 3) + \\ a\log_2(1/x_m + 1/x_m x_k + 1 + 2/x_m x_k) + 2a(\log_2 e)/e - \\ b(1/x'_m + 1/x'_m x'_k + 1 + 4/x'_m x'_k) - 1.5/(x'_k x'_m) - 1/x'_m \end{aligned} \quad (2.11)$$

For different values of the variables x *the left hand side of Inequality 2.11 is bounded from above.*

x_m	x'_m	x_k	x'_k	*Expression 2.11 <*
1	*1.4*	*1*	*2*	*-0.08*
		2	*4*	*-0.08*
		4	*10*	*-0.1*
		10	∞	*-0.1*
1.4	*1.9*	*1*	*2*	*-0.03*
		2	*4*	*-0.05*
		4	*10*	*-0.08*
		10	∞	*-0.1*
1.9	*2.4*	*1*	*2*	*-0.04*
		2	*5*	*-0.01*
		5	∞	*-0.02*
2.4	*3*	*1*	*2*	*-0.003*
		2	*4*	*-0.03*
		4	*10*	*-0.06*
		10	∞	*-0.1*
3	*3.5*	*1*	*2*	*-0.01*
		2	*5*	*-0.01*
		5	∞	*-0.02*
3.5	*4*	*1*	*2*	*-0.008*
		2	*5*	*-0.005*
		5	∞	*-0.02*

When $e \leq x_m$ the left hand side of Inequality 2.10 is less than or equal to

$$\begin{aligned} & 1/x_m(a\log_2(1/x_k + x_m + 1 + 2/x_k)) + \\ & 1/(x_k x_m)(a\log_2(x_k + x_m x_k + 3) + \\ & a\log_2(1/x_m + 1/x_m x_k + 1 + 2/x_m x_k) + \\ & 2a(\log_2 x_m x_k)/x_m x_k - \\ & b(1/x'_m + 1/x'_m x'_k + 1 + 2(\log_3 x'_k x'_m)/x'_m x'_k) - 1.5/(x'_k x'_m) - 1/x'_m \end{aligned} \tag{2.12}$$

The Expression 2.12 is bounded from above.

x_m	x'_m	x_k	x'_k	*Expression 2.12 <*
4	*4.3*	*1*	*2*	*-0.01*
		2	*8*	*-0.04*
		8	∞	*-0.2*
4.3	*4.9*	*1*	*2*	*-0.008*
		2	*15*	*-0.01*
		15	∞	*-0.3*
4.9	*5.7*	*1*	*2*	*-0.003*
		2	*15*	*-0.02*
		15	∞	*-0.2*
5.7	*6.8*	*1*	*2*	*-0.001*
		2	*15*	*-0.03*
		15	∞	*-0.2*
6.8	*8.5*	*1*	*2*	*-0.001*
		2	*15*	*-0.04*
		15	∞	*-0.2*
8.5	*11*	*1*	*2*	*-0.01*
		2	*15*	*-0.06*
		15	∞	*-0.2*
11	*18*	*1*	*2*	*-0.004*
		2	*15*	*-0.07*
		15	∞	*-0.2*
18	*80*	*1*	*2*	*-0.007*
		2	*15*	*-0.08*
		15	∞	*-0.2*
80	∞	*1*	∞	*-0.15*

Hence the inductive step is true for all cases.

Lemma 20 *After* $(\log_3 m)/2$ *iterations of the algorithm ATSP in a graph* $G_n \in \mathcal{G}$ *where* $n = 2m \geq 18$ *the cycle covers form a set* $\mathcal{K}$ *of cacti. The set has weight at least*

$$w(\mathcal{K}) \geq \frac{opt(G_n) - 1}{24}(\frac{\log_2 m}{4} - 1) - 2$$

Proof 20 *By Lemma 11 the minimum TSP tour in* G_n *has weight* $opt(G_n) < m + 1/16$. *Let the algorithm ATSP halt after* $(\log_3 m)/2$ *iterations. Then there are several cacti in* G_n. *We will show a lower bound of the weight of these cacti. First we show a lower bound for one cactus using the inequalities:*

$$w(K) + \delta(K) \geq w(K^S) + \delta(K^S) \geq l(K^S)a(\log_2 l(K^S) - bj)$$

Then we sum the weights of all cacti in the graph after $(\log_3 m)/2$ *iterations.*

By Lemma 17 we know that the first $\log_3 m$ *iterations of the algorithm ATSP on the graph* G_n *produces cycle covers with cycles of length two and three. Hence our cacti have only short cycles.*

We know that $w(K) \geq w(K^S)$ *and will show a lower bound for the cacti with the simple distance function. There are* m *large nodes in* G_n^S *(and in* G_n*), after* $j = (\log_3 m)/2$ *iterations there are several cacti connecting only large nodes and at most two cacti connecting a mix of large and small nodes. The number of nodes in a cactus,* $n(K)$ *is at least* $n(K) \geq 2^j = 2^{(\log_3 m)/2} = 2^{(\log_3 2 \log_2 m)/2} = m^{(\log_3 2)/2} \approx m^{0.32}$ *and* $n(K) \leq 3^j = 3^{(\log_3 m)/2} = \sqrt{m}$*, so* $m^{(\log_3 2)/2} \leq n(K^S) \leq \sqrt{m}$*.*

If $m \geq 4$ *then* $\sqrt{m} < m/2$ *and by Lemma 19* $w(K^S)+\delta(K) \geq l(K)(a \log_2 l(K) - bj)$*. For a cactus with only large nodes in* G_n^S*,* $l(K) = n(K^S) - 1$ *and certainly* $l(K) \geq n(K^S)/2$*. The weight of the cactus is* $w(K^S)+\delta(K^S) \geq l(K)(a \log_2 l(K) - bj) \geq (n(K^S)/2)(a \log_2(n(K^S)/2) - bj)$*. Let* $n(K_i^S) = m^{(\log_3 2)/2+d_i}$ *where* $0 \leq d_i \leq 1/2 - (\log_3 2)/2 \approx 0.18$*. The weight of a cactus* K_i^S *is*

$$
\begin{aligned}
w(K_i^S) + \delta(K_i^S) &\geq (m^{\log_3 2+d_i}/2)(a \log_2(m^{\log_3 2+d_i}/2) - bj) \geq \\
\frac{1}{2} m^{\log_3 2} m^{d_i} (\frac{1}{4}((\log_3 2 + d_i) \log_2 m - \log_2 2) &- \frac{24}{100} \log_3 m/2) \geq \\
\frac{1}{2}(\frac{1}{4} \log_3 2 \log_2 m - \frac{1}{4} &- \frac{24}{200} \log_2 m \log_3 2) m^{\log_3 2} m^{d_i} = \\
&(\log_2 m \log_3 2 \frac{26}{400} - \frac{1}{8}) m^{\log_3 2} m^{d_i}
\end{aligned}
$$

If $K_i^S, ..., K_r^S$ *are the cacti with no small nodes, then the total weight is:*

$$
\begin{aligned}
\sum_{i=1}^{r} w(K_i^S) - \delta(K_i^S) &\geq \\
\sum_{i=1}^{r} (\log_2 m \log_3 2 \frac{26}{400} - \frac{1}{8}) m^{\log_3 2} m^{d_i} - \sum_{i=1}^{r} \delta(K_i^S) &\geq \\
(\log_2 m \log_3 2 \frac{26}{400} - \frac{1}{8}) m^{\log_3 2} \sum_{i=1}^{r} m^{d_i} - (m + 1/16)/2 & \qquad (2.13)
\end{aligned}
$$

Since a cactus connecting large and small nodes has $n(K) < \sqrt{m}$*, the number of nodes in cacti with only large nodes is*

$$
\sum_{i=1}^{r} m^{\log_3 2+d_i} = m^{\log_3 2} \sum_{i=1}^{r} m^{d_i} \geq m - 2\sqrt{m}
$$

Inserted in Equation 2.13, if $m \geq 9$*:*

$$
\begin{aligned}
&\geq (\log_2 m \log_3 2 \frac{26}{400} - \frac{1}{8})(m - 2\sqrt{m}) - (m + 1/16)/2 \\
&\geq (opt(G_n) - 1)(\log_2 m \log_3 2 \frac{26}{400} - \frac{1}{8})(1 - 2/\sqrt{m}) - (1 + \frac{1}{16m}) \\
&\geq (opt(G_n) - 1) \frac{1}{8} (\frac{1}{4} \log_2 m - 1) \frac{1}{3} - 2
\end{aligned}
$$

The TSP tour

In the previous section we showed that the spanning cactus is heavy. Here we show that the TSP tour is heavy as well. In the algorithm by Frieze et al. there is no specification in which order the shortcuts are made. We assume that the TSP tour is built by a depth-first search, which is an efficient and natural implementation to make the shortcuts.

The level of a node $l(v)$ is the number of iterations it has in the subgraph, $l(v) = max\{i|v \in G_{n,i}\}$. In a TSP tour there is one edge *to* each node. We define an function from the edges in the TSP tour to cycles in the spanning cactus. The edge to the root of the cactus is not mapped to any cycle.

Definition 24 *Map an edge (v_j, v_i) to the last cycle v_i is in $(v_j, v_i) \rightarrow C, v_i \in C \in G_{n,l(v_i)}$.*

The next Lemma implies that the expected TSP tour in $\Omega(opt(G_n) \cdot \log n)$ – our main result.

Lemma 21 *A spanning cactus K is built by the algorithm ATSP on a graph $G_n \in \mathcal{G}$. The node for the next subgraph is chosen randomly with equal probability for all nodes in a cycle (row 11 in ATSP) and the TSP tour is built by a depth-first search starting in the last node in the subgraph. If $n = 2m \geq 18$ the expected length of the TSP tour is at least*

$$\frac{opt(G_n) - 1}{768}(\log_2 n - 5) - 1$$

Proof 21 *Let the algorithm halt after $(\log_3 m)/2$ iterations. Then there are several unconnected cacti in the graph, denote the cacti K. By Lemma 17 we know that the first $\log_3 m$ iterations of the algorithm ATSP on the graph G_n produces cycle covers with cycles with two or three nodes. Hence our cacti only have short cycles.*

Definition 24 is a mapping from the edges in the TSP tour to the cycles, C, in the cacti, K. Since there are several cacti there are several edges that are not mapped to any cycle.

$$E[\sum_{(v_j,v_i) \in TSP\ tour} d(v_j, v_i)] \geq \sum_C P[C\ in\ K] E[\sum_{(v_j,v_i) \rightarrow C} d(v_j, v_i)] \geq$$
$$\sum_C P[C\ in\ K] w(C)/8 = 1/8 \sum_C P[C\ in\ K] w(C) = E[w(K)]/8 \qquad (2.14)$$

The weight of the cacti is by Lemma 20

$$w(K) \geq \frac{opt(G_n) - 1}{24}(\frac{\log_2 m}{4} - 1) - 2$$

If Equation 2.14 holds the expected value of the TSP tour is at least larger than

$$\frac{opt(G_n) - 1}{768}(\log_2 n - 5) - \frac{1}{4}$$

Look at the 2-cycle (v_i, v_k), At iteration j the cycle connects two cacti $K_{i,j}$ and $K_{k,j}$. Suppose that they are connected in the given order. By Lemma 15 $K_{i,j}$ connects all nodes in the interval $[v_a, v_b]$ and $K_{k,j}$ connects nodes $[v_c, v_d]$.

Suppose that the node v_k is selected for the next iteration and that $l(v_k) = j+1$. The probability that v_k is in exactly one more iteration is at least $1/2$. The edge to *v_i in the TSP tour is the only edge mapping to the cycle. Since the node v_k is in exactly one more iteration there are two possibilities for the depth-first search at v_k. Either it chooses the edge (v_k, v_i) or it chooses an edge to a node in $[v_c, v_d]$ before it chooses the edge to v_i. In both cases the weight is $\geq d[v_i, v_c]$.*

Suppose that v_i is selected for the next iteration and that $l(v_i) = j+1$. Then the edge to *v_k is mapped to the cycle and the added distance is at least $d[v_b, v_k]$. Hence the expected value of the edge mapped to the cycle is $\geq 1/2(d[v_c, v_i]/2+d[v_b, v_k]/2) \geq d[v_i, v_k]/4 = w(C)/8$.*

Look at a triangle (v_i, v_k, v_l). At iteration j the nodes are roots two three cacti $K_{i,j}$, $K_{k,j}$ and $K_{m,j}$. The cacti are connected in the given order and connects the nodes $[v_a, v_b]$, $[v_c, v_d]$ and $[v_e, v_f]$. Suppose the cycle consists of the directed edges $(v_i, v_l), (v_l, v_k)$ and (v_k, v_i).

Suppose that v_l is selected for the next iteration and that $l(v_l) = j+1$. The edges to v_i and v_k are mapped to the cycle. At v_l the depth-first search either selects the edge (v_l, v_k) or a node in $[v_e, v_f]$ before v_k, in both cases the edge to v_k is at least $d[v_k, v_e]$. The only edge to v_i is from v_k hence the edge to v_i is at least $d[v_i, v_c]$. By equal probability v_k is selected for the next iteration. Suppose that $l(v_k) = j + 1$. Then the edges to v_i and v_l are mapped to the cycle. The edge to v_i is at least $d[v_i, v_c]$ and the edge to v_l is at least $d[v_b, v_l]$. If v_i is selected and $l(v_i) = j + 1$ the edge to v_l is at least $d[v_b, v_l]$ and the edge to v_k is at least $d[v_e, v_k]$. The expected value of the edges mapped to the cycle is $\geq 1/2((d[v_k, v_e] + d[v_i, v_c])/3 + (d[v_i, v_c] + d[v_b, v_l])/3 + (d[v_b, v_l] + d[v_e, v_k])/3) \geq d[v_i, v_l]/3 = w(C)/6$.

By symmetry the same arguments holds if the edges in the triangle changes direction. Hence Equation 2.14 holds.

2.5 To Conclude

For the deterministic assumptions on the algorithm, the graphs, G_n, form a family of asymmetric graphs for which the approximation algorithm for asymmetric TSP by Frieze et al., with our specifications, shows a worst case behaviour. The algorithm returns by Theorem 1 a TSP tour of weight greater than $(opt(G_n) \cdot \log_2 n)/(2 + o(1))$. The analysis of the algorithm by Frieze et al. shows that the algorithm gives a TSP tour with weight less than or equal to $opt(G_n) \cdot \log_2 n$, hence the analysis of the algorithm by Frieze et al. is tight up to a factor of $1/2$.

The ratio α between edges in different directions is greater than $n/2$ and the data dependent approximation algorithm by Frieze et al. is then proven to give an approximation better than $3\alpha/2 = 3n/4$ or $O(n)$. When we apply the data dependent algorithm to G_n, there are two possible outcomes. The algorithm converts the

asymmetric graph to a symmetric, uses the algorithm due to Christofides [7] and in the end arbitrarily chooses the direction of the found TSP tour. The undirected TSP tour is around the circle. If the direction is chosen to be clockwise the algorithm finds the optimum TSP tour of weight n. If the direction on the other hand is chosen to be anti-clockwise the directed cycle has weight $n \log_2 n$ which is the same as for the original approximation algorithm. Thus with one choice assumed to be bad the data dependent algorithm approximates the asymmetric TSP tour within a factor of $\log_2 n$. The expected approximation is $(\log_2 n)/2$ over the choice of orientation of the tour.

For the random assumption on the algorithm the graphs, G_n, are a family of symmetric graphs for which the algorithm by Frieze et al., shows a worst case behaviour. The algorithm returns by Theorem 2 a TSP tour with expected weight $\Omega(opt(G_n) \cdot \log n)$.

Chapter 3

Complexity of the Directed Spanning Cactus Problem

In this chapter we study the complexity of finding a spanning cactus in various graphs. First we show that the task of determining if there is a *directed spanning cactus* in a general unweighted digraph is **NP**-complete. The proof is a reduction from One-In-Three 3SAT. Secondly we show that finding the minimum spanning cactus in a directed, weighted complete graph with triangle inequality is polynomial time equivalent to finding the minimum TSP tour in the same graph and that they have the same hardness in approximation.

3.1 Introduction

In discrete mathematics, the *cactus* is a well-known graph structure and in *undirected* graphs they have been carefully studied. Cacti in directed graphs, though, have been much less studied.

Definition 25 *A strongly connected, directed graph where each edge is contained in at most (and thus, in exactly) one directed cycle is called a* directed cactus*.*

Definition 26 *A* spanning, directed cactus *for a directed graph G is a subgraph of G that is a directed cactus and connects all vertices in G.*

In undirected graphs finding the minimum cut in a graph is a well-known optimisation problem. Here a cactus is a useful and simple representation of the minimum cuts in a graph (there can be many). Cacti for this purpose are used for example by Fleischer in [11]. In 1994 Schaar [31] published a paper about Hamiltonian properties of *directed* graphs. He showed some results about graphs restricted to be directed cacti.

We study the complexity of finding a spanning, directed cactus in different types of graphs. First we show that the problem of finding a spanning cactus in a

general, unweighted, directed graph is **NP**-complete and then show that finding the minimum spanning cactus in a weighted, directed graph with triangle inequality is polynomial time equivalent to finding the minimum travelling salesman tour in the same graph.

Definition 27 *The* Spanning cactus problem (SCP) *is the problem of deciding, given a directed graph, if there is a spanning, directed cactus in the graph.*

Theorem 3 *SCP is* **NP***-complete.*

Corollary 1 *In a directed, complete graph with general distance function the minimum spanning cactus is* **NP***-hard to approximate within any factor.*

Corollary 1 can be shown with the same arguments as Sahni and Gonzalez [30] use to prove that it is impossible to approximate the minimum TSP tour within a constant factor in a general graph. Suppose that we can approximate the minimum spanning cactus within a factor r in a weighted complete graph then we can decide if there is a spanning cactus in a general directed graph in the following way: Give all edges weight 1, add edges of weight $r2|V|$ to make the graph complete. If there is a spanning cactus of weight less than $r2|V|$ there is a spanning cactus in the original graph, otherwise there is not. Since the original problem is **NP**-hard the corollary follows.

The Travelling Salesman Problem (TSP) is one of the most famous and well-studied combinatorial optimisation problems. We show that finding the minimum spanning cactus in a general, weighted digraph and finding the minimum TSP tour in the same graph are polynomial time equivalent problems. They also have the same hardness of approximation. Therefore a minimum spanning cactus is not directly useful for approximation algorithms of asymmetric TSP with triangle inequality.

Theorem 4 *Finding a spanning cactus of minimum total edge weight in an asymmetric, weighted, complete graph where the weights obey the triangle inequality is polynomial time equivalent to finding the minimum TSP tour in the same graph. They also have the same hardness of approximation.*

The well-known approximation algorithm for asymmetric TSP by Frieze, Galbiati and Maffioli [12] from 1982 builds a spanning cactus (which is not minimal) and then transforms it to a TSP tour. Their algorithm gives an approximation in $\log_2 n$. As a comparison the currently best approximation algorithm is by Kaplan, Lewenstein, Shafrir and Sviridenko [21] and gives an approximation of $3/4 \log_3 n < 0.842 \log_2 n$.

Notations and conventions

In a directed graph an edge from vertex A to vertex B is denoted AB, a path from A to B to C is denoted ABC and a cycle from A to B to C and back to A is denoted

ABCA. Considered cycles are always simple. When we study subgraphs (such as gadgets) we use the term *cactus branch*.

Definition 28 *Suppose there is a spanning cactus S in a directed graph G. In a subgraph $H \subseteq G$ the* cactus branch of S induced by H *is the set of edges* $\{e : e \in S \cap H\}$. *When it is clear what S and H we consider, we use the term* cactus branch.

Since a cactus is Eulerian the following well-known property of a directed cactus directly follows:

Lemma 22 *In a directed cactus every vertex has the same in- and out-degree.*

3.2 Proof that SCP is NP-complete

We will first show that SCP is in **NP** and then reduce One-In-Three 3SAT (which is an **NP**-complete problem [13, Problem LO4]) to SCP.

The definition of an **NP**-problem is that if the problem has a solution there is a witness convincing a polynomial time verifier that the problem is solvable. Our witness for SCP is the subgraph which we claim is a spanning cactus. The following algorithm determines in polynomial time if a subgraph of a directed graph is a spanning cactus.

Definition 29 *Spanning cactus check, SCC(G, S), is the following algorithm: Given a directed graph G and a subgraph S of G, if all three conditions below are true accept otherwise reject.*

- *The graph S is strongly connected.*
- *The graph S is spanning, i.e., every vertex of G is in S.*
- *The graph S is a cactus, i.e., the algorithm Cactus-check(S), defined below, accepts.*

Definition 30 *Cactus-check(S), is the following algorithm: Given a directed, strongly connected, graph S, accept if S is an empty graph or contains only one vertex. If the graph is not Eulerian reject, otherwise construct an Euler tour and traverse the edges in the given order. Push every visited vertex on a stack and mark the vertex as visited. If a vertex is marked as visited and is on the stack, pop all vertices above it (but not the vertex itself) from the stack. Continue until all vertices of the tour have been visited. If there is a vertex marked as visited, which is not on the stack, the test does not accept the graph as a cactus, otherwise it does.*

If the graph S is a cactus the algorithm Cactus-check recursively removes *drops* from S.

Definition 31 *A simple cycle where every vertex, except at most one, has in- and out-degree equal to one, $d^+(v) = d^-(v) = 1$, is called a* drop.

Lemma 23 *A cactus is either one simple cycle or a graph containing at least two drops.*

Proof 22 (Proof) *The proof is by induction over the numbers of simple cycles in the cactus. If a cactus has one cycle the statement is trivial and if a cactus has two simple cycles they are both drops. Suppose the lemma is true for a cactus with $k \geq 2$ cycles. To see that the lemma is true for a cactus with $k+1$ cycles proceed as follows: select an arbitrary cycle and remove it from the graph.*

If the selected cycle is a drop, the remaining graph has by assumption at least two drops. At most one drop was connected to the removed drop and therefore at least one drop in the reduced graph is a drop in the original graph. Thus the original graph has at least two drops; the removed one and at least one in the reduced graph.

If the selected cycle is not a drop it must be connected to at least two other cycles. Removing the cycle then divides the graph into at least two strongly connected components. If a component is one cycle it is a drop in the original graph. If a component has more than one cycle it has, by assumption, at least two drops. By the same argument as above at least one of them is a drop in the original graph. Thus every strongly connected component contributes with at least one drop in the original graph.

The argument in the lemma above shows that we can view a cactus as a tree of cycles where the drops are the leafs.

Lemma 24 *Let S be a directed, strongly connected graph with at least one drop and let S_0 be S with the drop removed. Then Cactus-check(S) = Cactus-check(S_0).*

Proof 23 *If S is not Eulerian then neither is S_0 and Cactus-check rejects both. Otherwise we can construct an Euler tour and traverse the nodes in the given order. When the algorithm reaches a drop, all vertices in the drop are pushed onto the stack and then all those vertices except the first one are immediately popped. Moreover the vertices in the drop will not appear later in the Euler tour. Thus if the algorithm rejects the graph S it will not reject because of any of the vertices in the drop and thus also rejects S_0. Similarly, if the algorithm accepts S it will also accept S_0.*

Lemma 25 *Given a directed, strongly connected graph S the algorithm Cactus-check(S) runs in polynomial time and accepts if and only if S is a cactus.*

Proof 24 *By Lemma 24 it is sufficient to consider S to be drop-free. A drop-free cactus is by Lemma 23, an empty graph and by definition the algorithm accepts such a graph.*

An Euler tour can be found in time $O(|E|)$ [8, Problem 22-3]. We prove that if the graph is not a cactus the algorithm rejects and then that if the algorithm rejects the graph is not a cactus. Let S be a drop-free graph which is not a cactus. The first cycle the algorithm pops from the stack is not a drop; therefore some popped vertex v in the cycle will appear later in the Euler tour. The algorithm might halt

and reject before v occurs again, otherwise it will find v which is visited but not on the stack, and reject.

If the algorithm rejects there is a visited vertex v which is not on the stack. The first edge from v in the Euler tour is then in a cycle which is popped from the stack. The Euler tour passes v twice and forms a cycle. This cycle is different from the first one since it is not popped from the stack. Hence the first edge from v in the Euler cycle is in two different cycles and the graph is not a cactus.

Lemma 26 *The algorithm SCC(G, S) determines, in polynomial time, if a given subgraph S of G is a cactus spanning G.*

Proof 25 (Proof) *A depth-first search from every vertex in S determines in polynomial time if S is strongly connected. If every vertex in G is in S the subgraph is spanning. The algorithm Cactus-check determines by Lemma 25 in polynomial time if the graph S is a cactus. Thus if all three conditions are true the graph is a spanning cactus.*

By Lemma 26 a verifier can check if the subgraph is a spanning cactus in polynomial time and the corollary trivially follows.

Corollary 2 *SCP is in* **NP**.

Reducing One-In-Three 3SAT to SCP

One-In-Three 3SAT is an **NP**-complete problem [13, Problem LO4]; by reducing One-In-Three 3SAT to SCP we show that SCP is **NP**-complete as well.

Definition 32 One-In-Three 3SAT *is the following decision problem: Given a set U of variables and a collection C of clauses over U such that each clause $c \in C$ has $|c| = 3$, is there a truth assignment for U such that each clause in C has exactly one true literal?*

Theorem 5 One-In-Three 3SAT *is* **NP***-complete even if no clause contains a negated literal [13, Problem LO4].*

The structure of the reduction is similar to the one Johnson and Papadimitriou use when they reduce Exact cover to Hamiltonian cycle [20] but there are more cases to cover since a cactus has more degrees of freedom than a Hamiltonian cycle. The variables and clauses from the One-In-Three 3SAT problem are represented by a graph. If and only if the graph contains a spanning cactus, One-In-Three 3SAT has a solution; furthermore the solution can be determined from the spanning cactus. The reduction is made in three steps. First we will construct the corresponding graph, then show that if there is a solution to One-In-Three 3SAT we can find a spanning cactus in the graph, and thereafter prove that if there is a spanning cactus in the graph we can find a solution to One-In-Three 3SAT.

Each clause in One-In-Three 3SAT is represented by a so called *clause-gadget* in the graph. A spanning cactus has three possible cactus branches in this gadget which correspond to the three possible assignments of the variables in the clause. Also each variable is represented by a gadget. There are two possibilities for the spanning cactus in the *variable-gadget* which correspond to the two values of a variable. To ensure consistency of the solution there are so called *xor-gadget* which connects a variable-gadget to the clause-gadget where the variable occurs.

A clause with variables $\{x_1, x_2, x_3\}$ is represented by a gadget as in Figure 3.1. Each variable corresponds to an edge in the gadget. If a *variable-edge* is in the spanning cactus the variable is false, otherwise it is true. Each variable is represented by a gadget as in Figure 3.2. The value of the variable is represented by two edges. Only one of the *value-edges* can be in the spanning cactus (Lemma 30) and, intuitively; if the *false-edge* is in the spanning cactus the variable is false and if the *true-edge* is in the spanning cactus the variable is true. All these gadgets are linked after each other in a cycle (Figure 3.3).

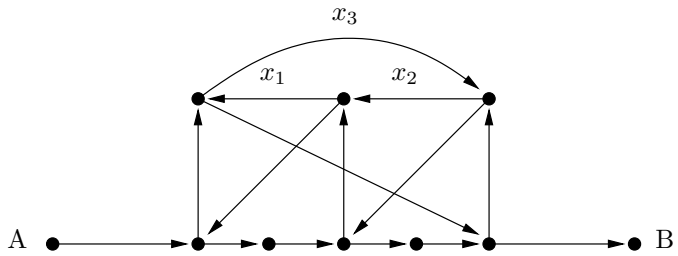

Figure 3.1: Clause-gadget.

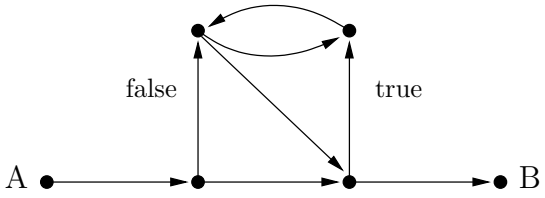

Figure 3.2: Variable-gadget.

To ensure that a spanning cactus gives a variable the same value in all clauses a variable-edge in an clause-gadget is connected to the true-edge in the variable-gadget by an xor-gadget as in Figure 3.4. The xor-gadget has the property that exactly one of the two edges it connects is in a spanning cactus (Lemmas 31 and 32). The inner structure of the xor-gadget is as in Figure 3.5. ABCD is the "true-edge"

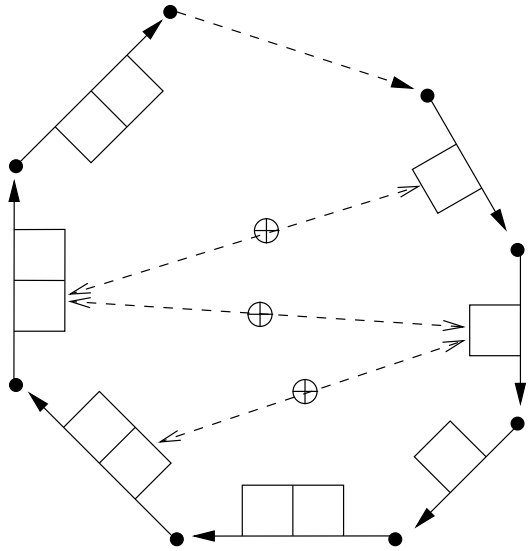

Figure 3.3: The structure of the graph. Clause-gadgets (left) and variable-gadgets (right) are linked together in a cycle. The variable-edges in the clause-gadgets are connected to the true-edges in the variable-gadgets by xor-gadgets. (Most of the xor-gadgets are omitted in the figure.)

in the variable-gadget and LKJI is the "variable-edge" in the clause-gadget. If one variable occurs in several clauses the xor-gadgets are linked together in the variable-gadget as in Figure 3.6. The figure shows two linked xor-gadgets but it can be extended to arbitrarily many. In Figure 3.6 AF is the true-edge in the variable-gadget, RO and VS are variable-edges in the clause-gadgets. In detail the linked xor-gadgets look like Figure 3.7.

If there is a solution to an instance of One-In-Three 3SAT we want it to be a spanning cactus in the constructed graph. We prove this by showing how to construct a spanning cactus from a solution of an instance of One-In-Three 3SAT.

Lemma 27 *Suppose that a graph is constructed from an instance of* One-In-Three 3SAT *as described above. If there is a solution to the instance of* One-In-Three 3SAT *then there is a spanning cactus in the constructed graph.*

Proof 26 *Suppose we have a satisfying assignment to an instance of* One-In-Three 3SAT*. A spanning cactus can then be constructed as follows: In the variable-gadgets let the value-edge with the same value as the variable be in the cactus. In the clause-gadgets let the two false variables be in the cactus. The xor-gadgets connects*

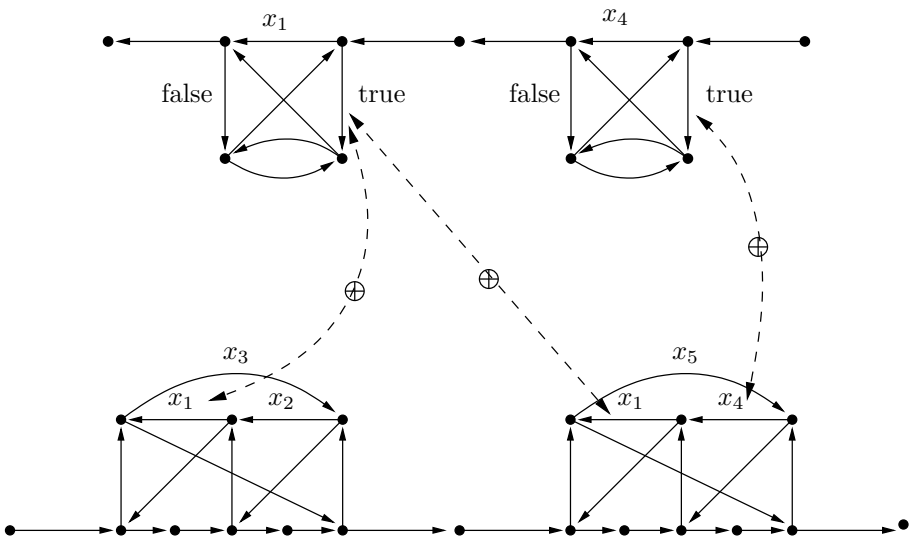

Figure 3.4: The xor connection between variable-edges in the clause-gadgets and true-edges in the variable-gadgets.

two "edges" where exactly one is in the cactus. We show which edges to choose in every gadget and that this is a spanning cactus.

In a satisfying assignment of the variables in a clause two variables are false and one is true. If a variable is false its variable-edge is in the spanning cactus otherwise it is not. Figure 3.8 shows three cactus branches which include exactly two of the three variable-edges in the clause-gadget. A variable is obviously true or false. The corresponding edge in the variable-gadget is in the cactus branch and we can find a cactus branch in the variable-gadget for each value of the variables (Figure 3.9). In a satisfying assignment a variable has a unique value and thus the xor-gadgets will only connect edges in the spanning cactus with edges that do not belong to the spanning cactus. Figure 3.10 shows cactus branches in the xor-gadget which includes exactly one of the two "edges" ABCD and LKJI.

The constructed subgraph is strongly connected: Choose two arbitrary vertices, they are in two gadgets since the graph only consists of gadgets. All variable- and clause-gadgets are connected and we can find a path between the vertices. If one vertex is in a xor-gadget it is connected to a variable-gadget or a clause-gadget and we can find a path to that vertex too.

One edge is in exactly one cycle: Let us first ignore the xor-gadgets and view the edges they connect as atomic edges. An edge in a variable- or clause-gadget is then in a cycle or in a path connecting the gadget to the rest of the graph. The cycles are not connected to any other part of the graph and every edge in the cycle is in that

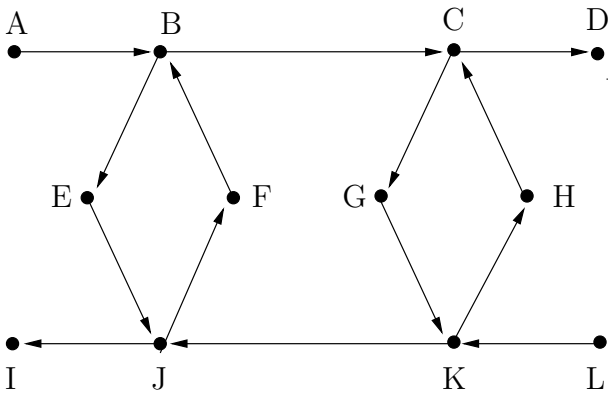

Figure 3.5: Xor-gadget. ABCD and LKJI are the edges which the xor-gadget connects.

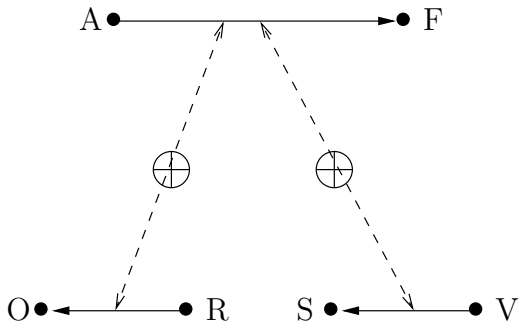

Figure 3.6: Linked xor-gadgets. If a variable occurs in two different clauses the xor-gadgets are linked. AF is the true-edge in the variable-gadget, RO and VS are variable-edges in the clause-gadgets.

cycle. The edges in the path is in the big cycle (Figure 3.3) but not in any other cycle. When we add xor-gadgets they replace an edge in the variable-gadget or the variable-edges in clause-gadgets, with a series of edges and diamonds *(Figure 3.10). But since exactly one of the "edges" the xor-gadget connects is in the cactus branch in an xor-gadget never connects a variable- and a clause-gadget. Therefore no edge is in more than one cycle.*

Hence if there is a solution of an instance of One-In-Three 3SAT *we can construct a spanning cactus in the corresponding graph.*

To complete the reduction, we prove that if there is a spanning cactus in our constructed graph there is a solution to One-In-Three 3SAT.

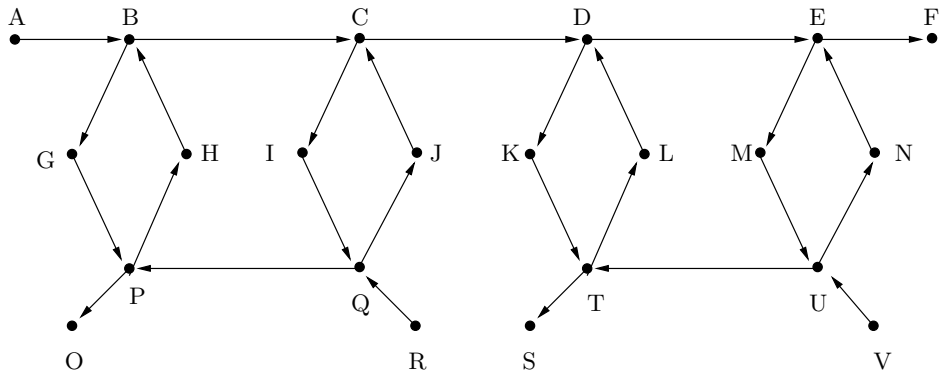

Figure 3.7: Linked xor-gadgets (the same as Figure 3.6)

Lemma 28 *Suppose that a graph is constructed from an instance of* One-In-Three 3SAT *as above. If there is a spanning cactus in our constructed graph, then there is a solution to the instance of* One-In-Three 3SAT*, furthermore the solution can be found via the spanning cactus.*

The proof proceeds by showing that any spanning cactus in the constructed graph defines a satisfying assignment to the instance of One-In-Three 3SAT. It is easy to see that the edges connecting the variable- and clause-gadgets are in the spanning cactus (Figure 3.3) since the spanning cactus is strongly connected. recall that some edges in the variable- and clause-gadgets are not really edges but xor-gadgets. Presently we view them as atomic edges and prove in Lemmas 31 and 32 that our view holds. Suppose there is a spanning cactus in our constructed graph. Then there is a cactus branch in every gadget. We will prove that every cactus branch corresponds to an assignment and that the assignment is consistent.

In the clause-gadget exactly two of the three variable-edges should be in the cactus branch, otherwise the cactus does not correspond to a satisfying assignment. The following lemma proves this and that the cactus branches in Figure 3.8 are the only possible cactus branches in the gadget.

Lemma 29 *Suppose that the clause-gadget (Figure 3.1) is a subgraph in an arbitrary graph. Vertices A and B are connected to the rest of the graph but no other vertices have any other edges than the ones in the figure. Any cactus branch corresponding to a spanning cactus includes exactly two of the edges x_1, x_2 and x_3.*

Proof 27 (Proof) *The path is restricted in several ways. It follows the lower horizontal edges to connect all vertices. If it traverses one vertical edge starting in v_i the cycle has to end in v_i to make the in- and out-degree equal (Lemma 22). The spanning cactus traverses exactly one of the vertical edges (otherwise one edge is*

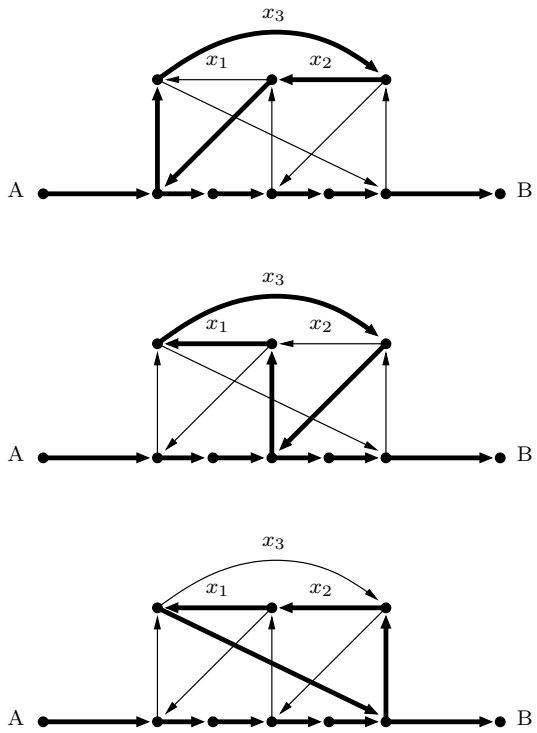

Figure 3.8: Possible cactus branches in an clause-gadget.

contained in more than one cycle). For each vertical line there is exactly one way to connect all vertices and to give all vertices an equal in- and out-degree (Figure 3.8).

A variable should, of course, have exactly one value. In other words, exactly one of the value-edges should be in the cactus branch. The following lemma proves this and Figure 3.9 shows the only possible cactus branches in the gadget.

Lemma 30 *Suppose a variable-gadget (Figure 3.2) is a subgraph in an arbitrary graph. Vertices A and B are connected to the rest of the graph but no other vertices have any other edges than the ones in the figure. Any cactus branch corresponding to a spanning cactus includes exactly one of the edges true and false.*

Proof 28 (Proof) *Since all vertices in a cactus have the same in- and out-degree (Lemma 22) there are only two possible ways to traverse the gadget (Figure 3.9).*

Recall that we introduced the xor-gadgets to ensure that the variable has the same value in all clauses. Specially we want xor-gadget to force that exactly one

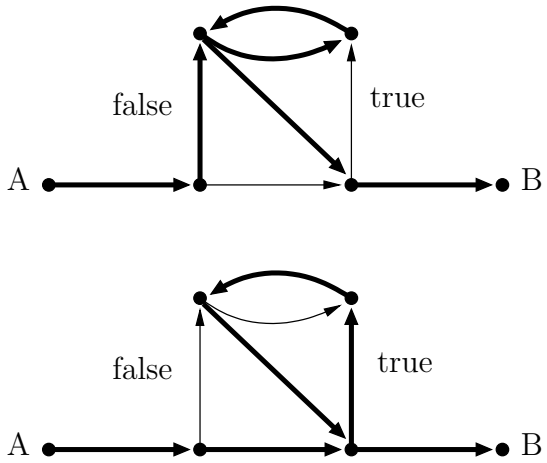

Figure 3.9: Possible cactus branches in a variable-gadget.

of the two edges it connects is in the spanning cactus. The following two lemmas proves this and that the cactus branches in Figure 3.10 are the only possible ones.

Lemma 31 *Suppose that the xor-gadget (Figure 3.5) is a subgraph in an arbitrary graph. Vertices A, D, I and L are connected to the rest of the graph but no other vertices have any other edges than the one in the figure. Any cactus branch corresponding to a spanning cactus contains either the edges AB and CD but not JI and LK or it contains JI and LK but not AB and CD.*

Proof 29 (Proof) *Since the spanning cactus is strongly connected the two* diamonds *(BEJFB and CGKHC) are in the spanning cactus. If the edge AB is in the cactus so are the edges BCD (Figure 3.10) since every vertex in a cactus has the same in- and out-degree (Lemma 22). For the same reason if the edge LI is in the cactus so are KJI (Figure 3.10). Hence at least one of ABCD and LKJI are in the spanning cactus.*

Assume that ABCD and LKJI are in the cactus. Then the edges BC and KJ are in the cactus and the diamonds *and the edges BC and KJ form three different cycles. The edges CG, GK, JF and FB are then contained in two cycles which contradicts the definition of a cactus.*

If one variable occurs in several clauses the xor-gadgets are linked together in the variable-gadget (Figure 3.7). Even for linked xor-gadgets Lemma 31 holds. More formally the Lemma can be extended to:

Lemma 32 *In an arbitrary graph two (or more) xor-gadgets linked as in Figure 3.7 form a subgraph. Single vertices as A, F, O, R, S and V (and possibly more) are*

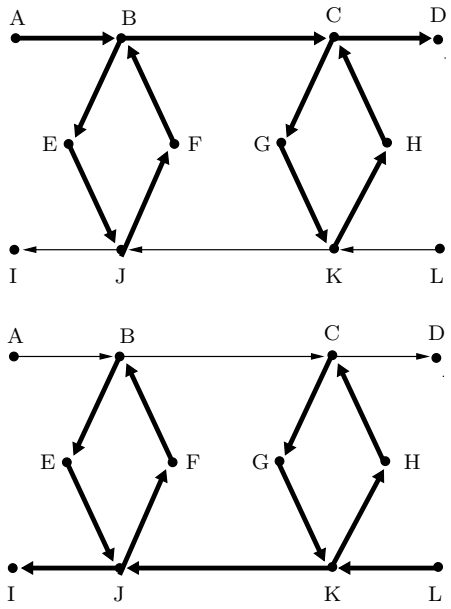

Figure 3.10: Cactus branches in a xor-gadget.

connected to the rest of the graph but no other vertices have any other edges than the ones in the figure. Any cactus branch corresponding to a spanning cactus either contains AB and EF or it contains RQ, PO, VU and TS (and possibly more).

Proof 30 (Proof) *All* diamonds *are in the spanning cactus since it is strongly connected. If the edge AB is in the cactus so are BCDEF (and possibly more) by the same argument as in Lemma 31. In the same way; if the edge RQ is in the cactus so are RQPO and if the edge VU is in the cactus so are VUTS (and possibly more).*

If the edges ABCDEF are in the cactus Lemma 31 proves that the edges RQ, PO, VU and TS can not be in the cactus.

If the edges RQPO are in the cactus we want to show that it the edges VUTS are in the cactus as well. If the edges RQPO are in the cactus the edges AB, BC, CD can not be in the spanning cactus by Lemma 31. If the edge CD is not in the cactus Lemma 31 shows that the edges VU and TS have to be in the cactus and that the edge EF can not be in the cactus. The argument can by induction be extended to arbitrary many xor-gadgets.

To conclude: If there is a spanning cactus in the graph every variable-gadget gives a value to the corresponding variable (Lemma 30). The construction of the xor-gadgets ensures that every variable has the same value in all clauses (Lemma 31).

Since there is a spanning cactus every clause has a satisfying assignment (Lemma 29). Thus we have a satisfying assignment of the instance of One-In-Three 3SAT and have proven Lemma 28.

Proof of the Main Theorem

We have constructed a graph from an instance of One-In-Three 3SAT and shown that if there is a satisfying assignment to the variables we can find a spanning cactus in the graph (Lemma 27). If there is a spanning cactus in the graph Lemma 28 shows that we can find a satisfying assignment via the spanning cactus. Thus if there is no satisfying assignment to the instance of One-In-Three 3SAT there is no spanning cactus in the constructed graph and vice versa. The result can be formalised to:

Theorem 6 *Suppose that a graph is constructed from an instance of* One-In-Three 3SAT *as described above. Then there is a spanning cactus in the constructed graph if and only if there is a satisfying assignment of the variables in the instance of* One-In-Three 3SAT*.*

SCP is in **NP** (Lemma 2) and the reduction from One-In-Three 3SAT to SCP can obviously be done in polynomial time. Since One-In-Three 3SAT is known to be **NP**-complete [13, Problem LO4], Theorem 6 proves that SCP also is **NP**-complete (Theorem 3) and we have shown our main result.

3.3 Asymmetric TSP and Spanning Directed Cactus

The Travelling Salesman Problem (TSP) is one of the most famous and well-studied **NP**-problems. It was proven **NP**-complete already by Karp [22] and it is in fact **NP**-complete for several special cases including Euclidean distance and Manhattan distance [20]. This means that an efficient algorithm for TSP is highly unlikely; hence it is interesting to investigate algorithms that compute *approximate* solutions. However Sahni and Gonzalez [30] showed that in the case of general distance functions it is **NP**-hard to find a tour even with weight within exponential factors of the optimum. When the distance function is symmetric and constrained to satisfy the triangle inequality the best known approximation algorithm is a factor 3/2-approximation algorithm due to Christofides [7]. To construct a TSP tour the algorithm finds a *minimum* spanning tree in the graph and then makes a minimum cost matching of vertices in the tree with odd degree. Together, the tree and the matching is an Eulerian graph. The Euler tour can, with short-cuts, be reduced to a TSP tour which obviously has weight less than or equal to the Euler tour.

The asymmetric case is much less understood. The twenty year old approximation algorithm, invented by Frieze, Galbiati and Maffioli [12], approximates the TSP tour within a factor of $\log_2 n$. The algorithm repeatedly makes minimum cycle covers of the graph and connects them to a spanning cactus (which is not minimal)

and then transforms the spanning cactus to a TSP tour. Despite a lot of effort in research during the last twenty years there are only some very recent algorithms which improves the constant factor of the approximation. The currently best algorithm is by Kaplan, Lewenstein, Shafrir and Sviridenko [21]. Their algorithm decomposes multigraphs and gives an approximation of $3/4 \log_3 n < 0.842 \log_2 n$. There is only a miniscule lower bound: Papadimitriou and Vempala [27] recently proved that it is **NP**-hard to approximate the minimum TSP tour within a factor less than $220/219 - \epsilon$, for any constant $\epsilon > 0$. Hence, any algorithm approximating the minimum TSP tour in an asymmetric graph within a factor independent of the number of vertices n is of great interest to the community.

In order to construct such an algorithm it is natural to try to generalise the ideas used by Christofides [7]. In particular, it seems fruitful to search for structures similar to that of a spanning tree in asymmetric graphs. One such structure is the spanning cactus. We observe however, that finding the minimum spanning cactus and the minimum TSP tour in an asymmetric weighted complete graph are polynomial time equivalent problems. They also have the same hardness of approximation. Therefore it can not be easier to find a minimum spanning cactus than a minimum TSP tour.

Proof 31 (Proof of Theorem 4) *The TSP tour is a spanning cactus and therefore the weight of the minimum spanning cactus is less than or equal to the TSP tour's weight.*

If we have a minimum spanning cactus it is possible to transform it into a TSP tour in the following way: Start in an arbitrary vertex, traverse the spanning cactus in the order of an Euler tour. If an edge goes to an already visited vertex replace the edge to the vertex and the next edge in the Euler tour with the edge short-cutting them. If the new edge goes to a visited vertex repeat until an unvisited vertex is found or to the end of the Euler tour. The triangle inequality guarantees that the weight of the short-cut edge is less than or equal to the combined weight of the original edges. The found TSP tour therefore has a weight less than or equal to the minimum spanning cactus weight.

Secondly, we prove that TSP can be approximated within c if and only if the size of the spanning cactus can be approximated within c. Every TSP tour is a spanning cactus and hence a c-approximation algorithm for TSP approximates the minimum spanning cactus within the same ratio. Conversely, a c-approximation algorithm for the minimum spanning cactus can be used to construct a c-approximate TSP tour by the construction outlined in the previous paragraph.

Chapter 4

Symmetric Representation of Asymmetric Graphs with an Application to Asymmetric TSP

This chapter is based on joint work with Lars Engebretsen, and my contribution is approximately 50%.

An instance of the *Asymmetric TSP with triangle inequality* can be transformed into an instance of the *Symmetric TSP* in a certain related bipartite graph. We show, that, in spite of the fact that the edge weights in this bipartite graph do not obey the triangle inequality, the weights of minimum TSP tours in the two corresponding instances are within constant factors of each other. The symmetric bipartite realization of instances of Asymmetric TSP therefore open up new possibilities to construct both stronger approximation algorithms for the problem as well as stronger approximation hardness results.

4.1 Introduction

A directed graph can be represented in a natural way by an undirected bipartite graph as follows: Every vertex in the directed graph is represented by two copies of the vertex—one "source" copy and one "sink" copy; every arc (u, v) in the directed graph is represented as an undirected edge between the "source" copy of u and the "destination" copy of v in the bipartite graph. Under the above transformation, certain structures in the bipartite graph correspond to related structures in the asymmetric graph: For instance, a perfect matching in the bipartite graph corresponds to a cycle cover in the directed graph. This is the main idea underlying the polynomial time algorithm that computes a minimum-weight cycle cover of a directed graph—this algorithm in fact computes a minimum-weight maximal matching in the corresponding bipartite graph.

This note considers similar relations between other structures in asymmetric and corresponding bipartite graphs. As an example, we consider Hamiltonian cycles and Eulerian subgraphs of directed graphs and prove that they correspond to subgraphs with certain structure in the corresponding bipartite graph. Although this combinatorial problem has an interest of its own, we believe that our main result is a characterization of the *Asymmetric TSP with triangle inequality* that has potential applications in the field of approximation algorithms and approximation hardness.

TSP with triangle inequality, i.e., instances of TSP where the distances between the cities satisfy the triangle inequality, is a classical combinatorial optimization problem. The decision version of this special case was shown to be **NP**-complete already in Karp's original 1972 paper on **NP**-completeness [22], which means that we have little hope of computing exact solutions in polynomial time. Christofides [7] famous algorithm from 1976 finds a TSP tour with cost within a factor of 3/2 from the cost of an optimum tour for distance functions that are *symmetric*. The so called *Asymmetric TSP with triangle equality* is, however, far less understood—it is in fact one of the most notorious open problems with respect to approximability. Frieze, Galbiati and Maffioli [12] constructed an algorithm in 1982 that approximates the optimum within $\log_2 n$, where n is the number of cities. Today, more than 20 years later, the best known algorithm [21] still guarantees only a logarithmic approximation ratio and the strongest known approximation hardness result [27] states that it is **NP**-hard to approximate the optimum within a factor of $1 + \epsilon$ for some ϵ which is roughly 10^{-2}. Obviously, any new insight into the structure of the asymmetric TSP with triangle inequality is of considerable interest to the community.

In this note, we present an alternate view of asymmetry in TSP instances. Specifically, we construct from an instance of the asymmetric TSP an instance of the *symmetric TSP* in a certain bipartite graph and then prove, that, when the distance function in the original asymmetric TSP instance obeys triangle inequality, the cost of the optimum tours in the two instances are within constant factors from each other. This implies that in order to construct a constant-factor approximation algorithm for the asymmetric TSP with triangle inequality, it is enough to construct such an algorithm for the symmetric bipartite graphs mentioned above. Alas, the metric in those bipartite graphs do not obey the triangle inequality. TSP with general, symmetric, distance functions that do not obey the triangle inequality cannot be approximated even within exponential factors. However, "our" bipartite graphs have a special structure and we believe that they could potentially provide an alternate way to attack the problem of devising algorithms for the asymmetric TSP with triangle inequality.

The currently strongest polynomial time approximation algorithms for asymmetric TSP with triangle inequality all use the idea of *subtour patching*, which essentially amounts to computing a sequence of cycle covers of the weighted directed graph describing the instance and then using these covers to form an Eulerian subgraph of the given instance. It seems, however, that it is difficult to use this approach to push the approximation ratio to $o(\log n)$. The bipartite symmetric

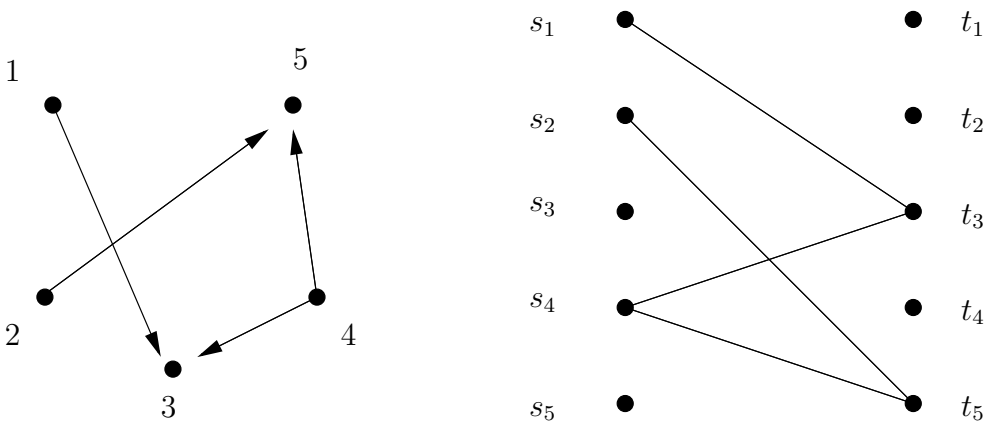

Figure 4.1: **Left**: An asymmetric graph G. **Right**: The bipartite realization $B(G)$.

version of the problem may be more susceptible to other algorithmic techniques that give better approximation ratios. We have not been able to successfully use the approach ourselves, but we hope that this note will enable other researchers to continue our efforts.

4.2 Bipartite Realization of Asymmetric Graphs

Intuitively, the bipartite realization of a directed graph G contains two copies of every vertex in G, one "source vertex" and one "destination vertex". An arc (u, v) in G is then represented as an undirected edge from the source vertex corresponding to u to the destination vertex corresponding to v (Fig. 4.1).

Definition 33 *Given an asymmetric unweighted graph G with vertex set V and arc set $A \subseteq V \times V$, the* bipartite realization of G*, denoted by $B(G)$, is defined as the graph with vertex set $\{s_v : v \in V\} \cup \{t_v : v \in V\}$ and edge set $\{\{s_u, t_v\} : (u, v) \in A\}$.*

For a weighted asymmetric graph G, the bipartite realization is defined as above with the addition that the weight of the edge $\{s_u, t_v\}$ in $B(G)$ is defined as the weight of the arc (u, v) in G.

By definition, G and $B(G)$ always have the same weight. Doing the construction described in Definition 33 "backwards", one can construct an asymmetric graph on, say, n vertices from a given bipartite graph on $2n$ vertices if the bipartition has well-defined "source" and "destination" sides that contain n vertices each. Formally: If G_B is a symmetric bipartite graph where the two sides in the bipartition have the same size and there is a designated "source" side, there is a unique asymmetric graph G such that $B(G) = G_B$.

It is easy to see that self-loops in an asymmetric graph translates to edges of the form $\{s_v, t_v\}$ in the bipartite realization and that Definition 33 also works for multigraphs: the bipartite realization of an asymmetric multigraph is a symmetric bipartite multigraph. In the rest of this paper, however, we only consider simple graphs. Consequently, there are no edges of the form $\{s_v, t_v\}$ in the bipartite graphs we consider.

The bipartite realization described in Definition 33 can be applied not only to entire graphs but also to subgraphs. Given an asymmetric graph G and some subgraph H of G, the bipartite realization of H, denoted by $B(H)$, is a subgraph of $B(G)$. The remainder of this note considers relationships between H and $B(H)$. In particular, we prove results of the form "if H has a certain structure with respect to G, then $B(H)$ has a certain structure with respect to $B(G)$" and vice versa. As an application, we use these relationships to prove that if G is an instance of the asymmetric TSP with triangle inequality, the cost of a minimum TSP tour in G and the cost of a minimum TSP tour in $B(G)$—which is a symmetric instance without triangle inequality—are within constant factors of each other.

We first study connectivity. It is easy to see that the bipartite realization of a connected asymmetric graph need not be connected: Consider the graph with vertex set $\{u, v\}$ and arc set $\{(u, v)\}$. However, the bipartite realization of a connected asymmetric graph has another property:

Definition 34 *Let G be an asymmetric graph with vertex set V. The bipartite realization $B(G)$ is* overlapping *if it holds that there is, for every non-empty proper subset F of vertices in $B(G)$ such that $|F \cap \{s_v, t_v\}| \neq 1$ for all $v \in V$, at least one edge in $B(G)$ that connects a vertex in F with a vertex outside F.*

Lemma 33 *The bipartite realization of a connected asymmetric graph is overlapping. Conversely, a bipartite realization that is overlapping corresponds to a connected asymmetric graph.*

Proof 32 *It can be seen that the requirement on F stated in Definition 34—for every subset F of vertices in $B(G)$ such that $|F \cap \{s_v, t_v\}| \neq 1$ for all $v \in V$—can be equivalently stated: for every subset F of vertices in $B(G)$ that can be written as $\{\{s_u, t_u\} : u \in U\}$ for some nonempty $U \subset V$.*

Let G be an asymmetric graph. If the graph G is connected then for every nonempty set $U \subset V$ there is either an arc (u, v) for some $u \in U$ and some $v \in V \setminus U$ or an arc (v, u) for some $u \in U$ and some $v \in V \setminus U$. Hence, there is an edge connecting a vertex in the corresponding set $F_U = \{\{s_u, t_u\} : u \in U\}$ with a vertex outside F_U. To conclude, $B(G)$ is overlapping in this case.

Conversely, if $B(G)$ is overlapping, then for every nonempty set $U \subset V$ it holds that there is at least one edge between a vertex in the set $F_U = \{\{s_u, t_u\} : u \in U\}$ and a vertex outside F_U. This implies that there is, for every nonempty set $U \subset V$ an arc from/to a vertex in U to/from a vertex in $V \setminus U$, i.e., that G is connected.

For the special case when the asymmetric graph is a Hamiltonian cycle, the bipartite realization has even more structure.

Lemma 34 *Let G be an asymmetric graph and H be a Hamiltonian cycle in G. Then $B(H)$ is an overlapping bipartite perfect matching in $B(G)$. Conversely, any overlapping perfect matching in $B(G)$ is the bipartite realization of a Hamiltonian cycle in G (Fig. 4.2).*

Proof 33 *A Hamiltonian cycle has in-degree one and out-degree one at every vertex. Therefore, it follows from the construction in Definition 33 that $B(H)$ has degree one at every vertex, i.e., it is a matching. Since a Hamiltonian cycle visits every node, the matching is perfect. Finally, since a cycle is connected, Lemma 33 implies that the matching is overlapping.*

Conversely, let H' be an overlapping perfect matching in $B(G)$ and consider the following walk in G: From an arbitrary vertex v, walk to a vertex w such that $\{s_v, t_w\}$ is an edge in H'; Repeat until the walk returns to a visited vertex. Since H' is a perfect matching, there is, for every v, a unique w such that $\{s_v, t_w\}$ is an edge in H'. Therefore the walk is well-defined. By the pigeon hole principle, the walk eventually returns to a visited vertex. We claim that it has then visited all vertices.

Let V denote the vertex set of G and suppose that the walk returns having visited some set $U \subseteq V$ of vertices in G. Let $F_U \subseteq H'$ be the set of edges are incident to any of the vertices $\{s_u : u \in U\} \cup \{t_u : u \in U\}$. Now, there can be no edge $\{s_u, t_v\}$ such that $u \in U$ and $v \notin U$ since this would imply that the walk must have visited $v \notin U$. Similarly, there can be no edge $\{s_v, t_u\}$ such that $u \in U$ and $v \notin U$. Hence, there are no edges connecting a vertex in F_U with a vertex outside F_U. Since H' is overlapping, Definition 34 implies that either $F_U = \emptyset$ or else $F_U = H'$. Since U is non-empty, however, also F_U must be non-empty. So the only remaining possibility is that $F_U = H'$, which, in turn, implies that $U = V$.

We have seen above that a Hamiltonian cycle in G corresponds to a perfect matching in $B(G)$. We now investigate what a Hamiltonian cycle in $B(G)$ corresponds to in G.

Lemma 35 *Let G be an asymmetric graph. A Hamiltonian cycle in $B(G)$ is the bipartite realization of an Eulerian subgraph of G. Moreover, every vertex in this subgraph has in-degree two and out-degree two (Fig. 4.3).*

Proof 34 *Let H be such that $B(H)$ is a Hamiltonian cycle in $B(G)$. Since a Hamiltonian cycle is connected and visits every vertex, $B(H)$ is overlapping; hence Lemma 33 shows that H is connected. Since every vertex in $B(H)$ has degree two, Definition 33 implies that H has in-degree two and out-degree two at every vertex. Thus H is Eulerian.*

The reverse of Lemma 35 does, in fact, not hold—a counter-example is shown in Fig. 4.4. However, it can be seen that for every Eulerian subgraph H of G with the additional property that every vertex has in-degree two and out-degree two, $B(H)$ is a *cycle cover of* $B(G)$, i.e., a subgraph of $B(G)$ where every vertex has degree two, that is overlapping.

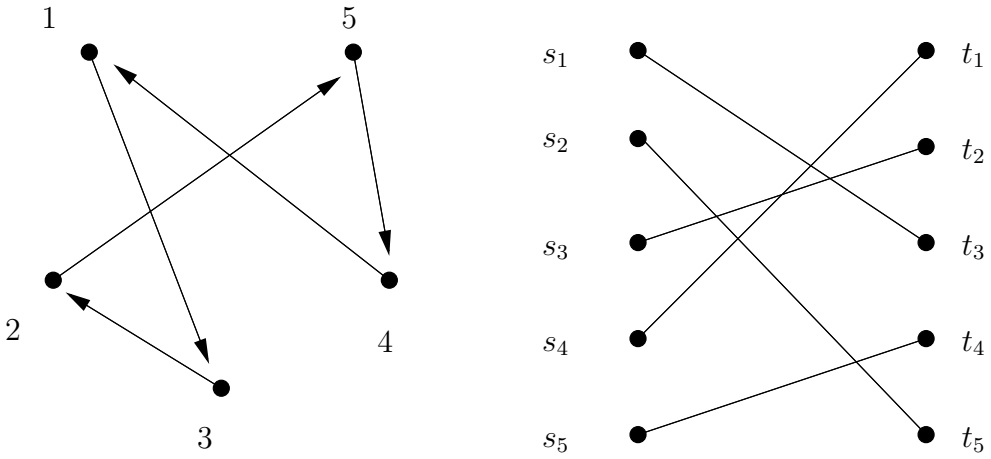

Figure 4.2: **Left**: An asymmetric Hamiltonian cycle H. **Right**: The bipartite realization $B(H)$—an overlapping perfect matching.

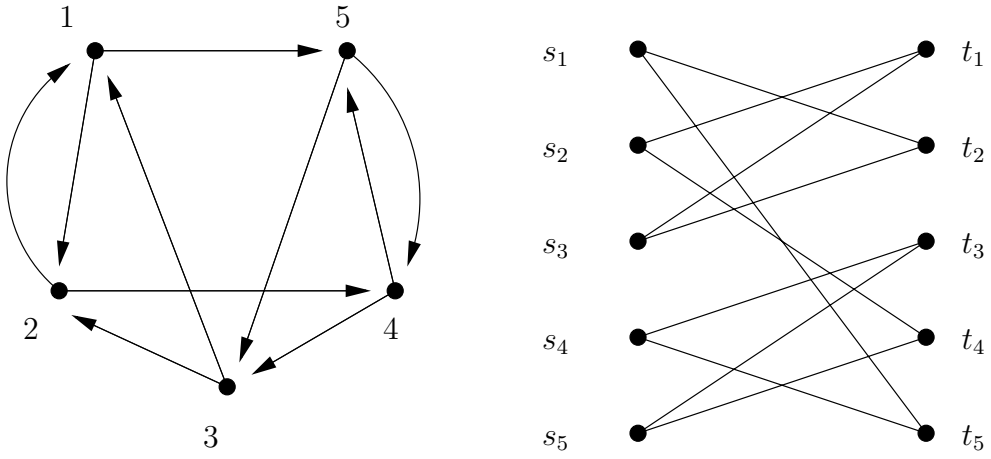

Figure 4.3: **Left**: An asymmetric Eulerian graph H with in-degree two and out-degree two at every vertex. **Right**: The bipartite realization $B(H)$—a Hamiltonian cycle.

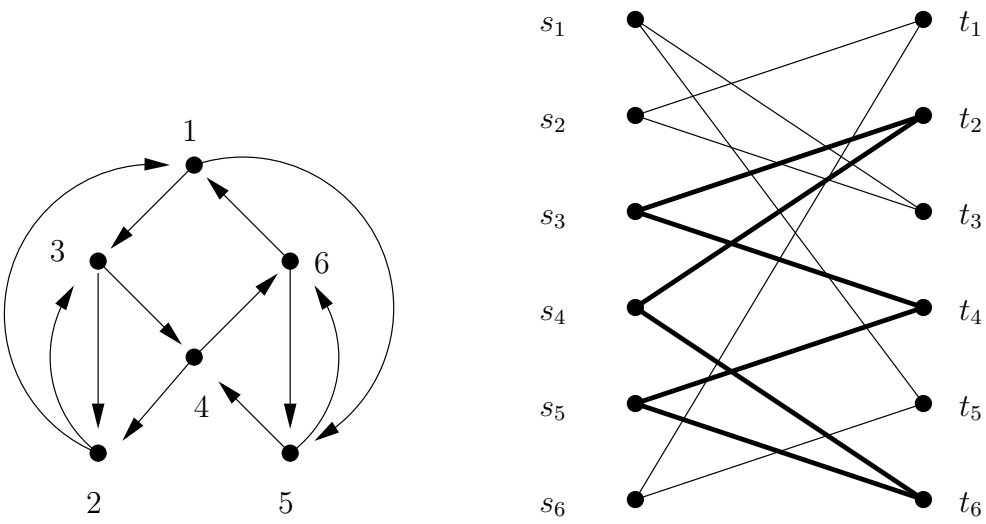

Figure 4.4: **Left**: An asymmetric Eulerian graph G with in-degree two and out-degree two at every vertex. **Right**: The bipartite realization $B(G)$—an overlapping cycle cover.

Lemma 36 *Let G be an asymmetric graph and H be an Eulerian subgraph of G with in-degree two and out-degree two at every vertex. Then $B(H)$ is an overlapping cycle cover of $B(G)$. Conversely, any overlapping cycle cover of $B(G)$ is the bipartite realization of an Eulerian subgraph of G with in-degree two and out-degree two at every vertex (Fig. 4.4).*

Proof 35 *By Definition 33 every vertex in $B(H)$ has degree two, hence $B(H)$ is a cycle cover. Since H is connected, it follows from Lemma 33 that $B(H)$ is overlapping.*

Conversely, let H' be an overlapping cycle cover of $B(G)$ and let H be such that $B(H) = H'$. Since every vertex in H' has degree two, it follows that every vertex in H has in-degree two and out-degree two. Since H' is overlapping, Lemma 33 implies that H is connected.

We remark that Lemma 33 shows that it is critical that the cycle cover H' in the proof above is overlapping for the corresponding asymmetric subgraph H to be connected.

4.3 Application to Asymmetric TSP

Theorem 7 *Let G be a complete, weighted, asymmetric graph and suppose that the weights satisfy the triangle inequality. Denote by $w(T_A)$ the weight of a minimal*

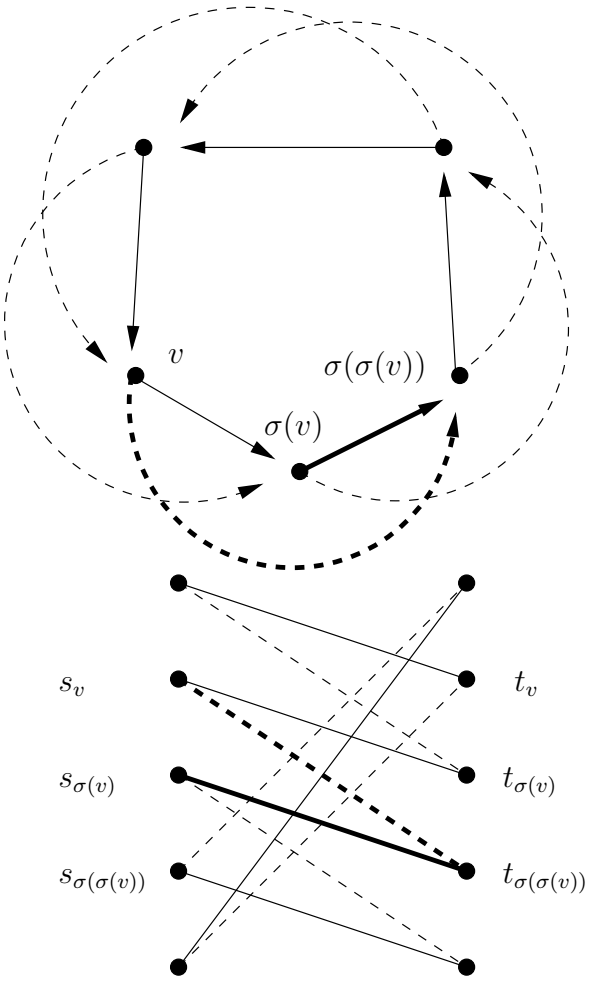

Figure 4.5: **Left**: An extended, asymmetric TSP tour H. **Right**: The bipartite realization $B(H)$—a Hamiltonian cycle.

TSP tour in G and by $w(T_B)$ the weight of a minimal TSP tour in $B(G)$. Then it holds that $w(T_A) \leq w(T_B) \leq 3w(T_A)$, or, equivalently, that $\frac{1}{3}w(T_B) \leq w(T_A) \leq w(T_B)$.

Proof 36 *To see that $w(T_A) \leq w(T_B)$, note that the minimum TSP tour in $B(G)$ is a Hamiltonian cycle; by Lemma 35 the corresponding subgraph in G is Eulerian and has the same weight. An Eulerian graph can be converted to a TSP tour by shortcuts. Since the edge weights in G obey the triangle inequality the constructed TSP tour has weight less than or equal to the weight of the minimum TSP tour in $B(G)$. Hence there exists a TSP tour in G with weight at most $w(T_B)$.*

To see that $3w(T_A) \geq w(T_B)$, consider an arbitrary minimum TSP tour in G. Denote by V the vertex set of G and let $\sigma: V \to V$ denote the successor of a vertex in the TSP tour, i.e., the TSP tour contains the arcs $\cup_{v \in V}(v, \sigma(v))$. Let H denote the subgraph of G that contains the edges

$$\left(\bigcup_{v \in V} \big(v, \sigma(v)\big)\right) \cup \left(\bigcup_{v \in V} \big(v, \sigma(\sigma(v))\big)\right).$$

In words, H contains the TSP tour plus "jumps" of length two in the tour (see Fig. 4.5). Since the edge weights in G satisfy the triangle inequality, the weight of $(v, \sigma(\sigma(v)))$ is at most the weight of $(v, \sigma(v))$ plus the weight of $(\sigma(v), \sigma(\sigma(v)))$; therefore the weight of H is at most $3w(T_A)$. Since H is Eulerian with in-degree two and out-degree two at every vertex, Lemma 36 shows that $B(H)$ is an overlapping cycle cover of $B(G)$. We now prove that this cycle cover consists of only one cycle, thereby establishing that a minimum TSP tour in $B(G)$ has weight at most $3w(T_A)$.

Since $(v, \sigma(v))$ is an arc in the considered minimum TSP tour in G, it follows from the construction of H described above that the edges $\{s_v, t_{\sigma(\sigma(v))}\}$ and $\{t_{\sigma(\sigma(v))}, s_{\sigma(v)}\}$ form a path of length two in $B(H)$. Since σ defines a permutation of the vertices in G, the path

$$\bigcup_{v \in V} \big(\{s_v, t_{\sigma(\sigma(v))}\} \cup \{t_{\sigma(\sigma(v))}, s_{\sigma(v)}\}\big)$$

is a cycle in $B(H)$. This cycle has length $2|V|$ and contains all vertices of $B(G)$.

Intuitively, the second part of the proof above defines a walk in $B(H)$ that corresponds to a "walk" in H that alternates between following a "long" edge forwards and a "short" edge backwards as indicated in Fig. 4.5.

Chapter 5

Finding a Perfect 2-Matching without 6-Factors in a Bipartite Graph is NP-complete

In a symmetric bipartite graph all cycles has even length and the shortest cycle has length four. For an arbitrary symmetric graph there is a polynomial time algorithm that finds the minimum cycle cover by matching. In 1999 Hartvigsen [16] give a polynomial time algorithm which finds square-free 2-matchings in bipartite graphs. For arbitrary graphs Papadimitriou has shown that it is **NP**-complete to find a perfect 2-matching without cycles of length five or less [9]. It has been conjectured [15] that finding a 2-matching without cycles of length six or less in a bipartite graph is **NP**-hard but no proof has been published. Here we give a proof and hence close the gap for bipartite graphs.

Theorem 8 *To determine if a graph has a perfect 2-matching with no polygon of size five or less is* **NP***-complete [9].*

Definition 35 *B_6 is the following decision problem: Given a symmetric bipartite graph, is there a perfect 2-matching such that there are no cycles of length six or less in the matching?*

We will show the following:

Theorem 9 *B_6 is* **NP***-complete.*

Corollary 3 *For any $k \geq 6$ B_k is* **NP***-complete.*

5.1 Proof of NP-completeness

B_6

The problem B_6 is obviously in **NP**. One-In-Three 3SAT is an **NP**-complete problem [13, Problem LO4]; by reducing One-In-Three 3SAT to B_6 we show that B_6 is **NP**-complete as well.

Definition 36 One-In-Three 3SAT *is the following decision problem: Given a set U of variables and a collection C of clauses over U such that each clause $c \in C$ has $|c| = 3$, is there a truth assignment for U such that each clause in C has exactly one true literal?*

Theorem 10 One-In-Three 3SAT *is* **NP***-complete even if no clause contains a negated literal [13, Problem LO4].*

Given an instance of One-In-Three 3SAT with an consistent assignment to the variables we can construct an instance of B_6 in the following way:

For every variable $x_i \in U$ we there is a *variable gadget* (Figure 5.1). There are two possible 2-matchings in the variable gadget; one containing the edge e_{true} and the other containing the edge e_{false}. If the variable is true the edge e_{true} is in the matching, and contrary if the variable is false the edge e_{false} is in the matching.

In every gadget the vertices are filled or not filled. Since no edge connect vertices of the same "colour" the graphs are bipartite.

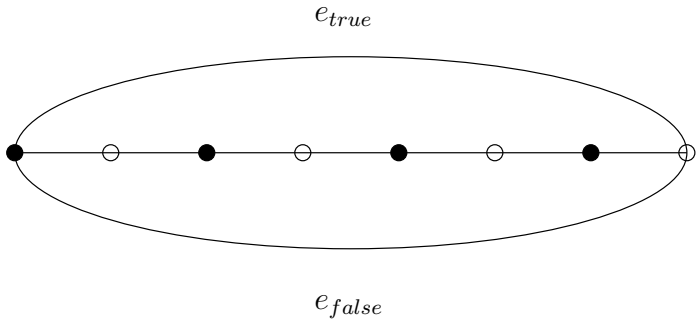

Figure 5.1: The variable gadget.

For every clause $c = \{x_i, x_j, x_k\} \in C$ there is a *clause gadget* (Figure 5.2). We associate each variable in the clause with an edge in the clause gadget. If a variable x_i is true the edge x_i is not in the perfect 2-matching and otherwise it is in the matching. Since this three edges connects one vertex it is obvious that exactly one of them is true in any perfect 2-matching.

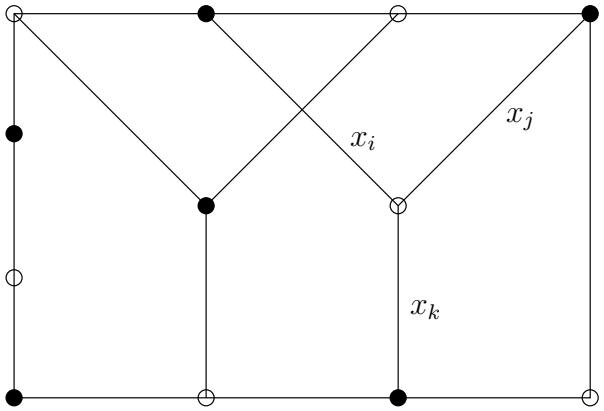

Figure 5.2: The clause gadget.

To give a variable x_i a consistent value in every clauses we need an *xor gadget* (Figure 5.3). The edges e'_{true} and e''_{true} replaces the edge e_{true} in the variable gadget and the edges x'_i and x''_i replaces the edge x_i in the clause gadget. A perfect 2-matching without cycles of length six or less contains either the edges e'_{true} and e''_{true} or the edges x' and x''. If a variable is in several clauses the xor gadgets are linked after one other.

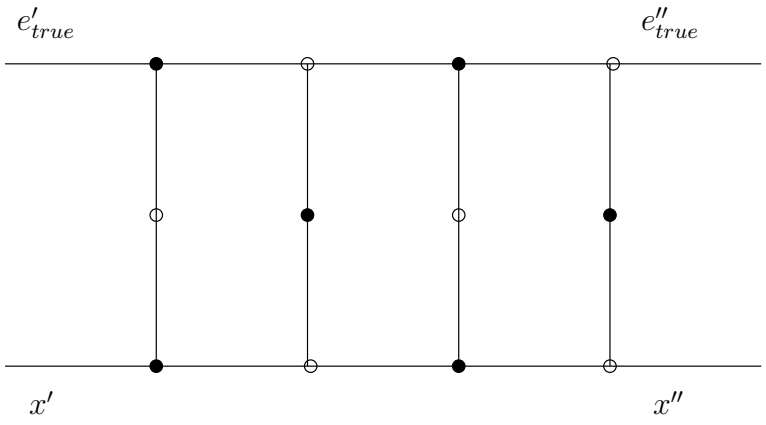

Figure 5.3: The xor gadget.

To make sure that every variable x_i in a clause can be true. Figure 5.4, 5.5 and 5.6 show the three possible perfect 2-matchings without cycles of length six

or shorter in the clause gadget. Remember that an edge x_i is one side in the xor gadget which consists of five edges, therefore the cycle with the edges x_j and x_k in Figure 5.4 has length larger than six.

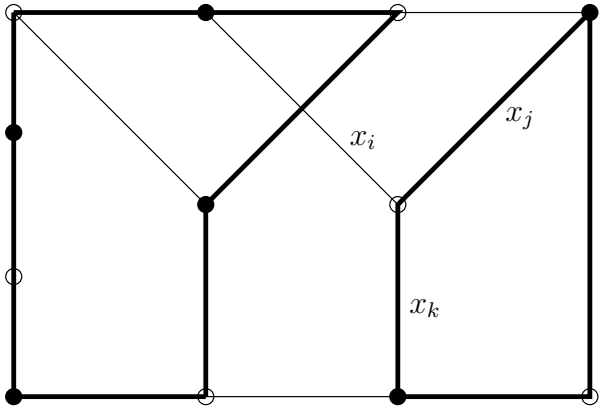

Figure 5.4: A perfect 2-matching excluding the edge x_i.

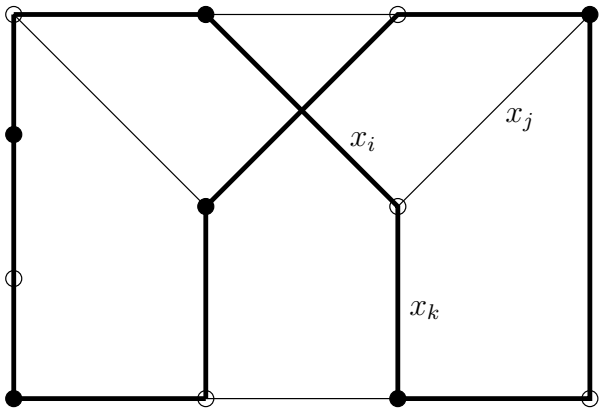

Figure 5.5: A perfect 2-matching with no cycles of length six and excluding the edge x_j.

The constructed graph contains a perfect 2-matching without cycles of length six or shorter.

Suppose there is a perfect 2-matching without cycles of length six or shorter in the constructed graph. We will show that there is a consistent solution to the instance of One-In-Three 3SAT and that we can extract the solution.

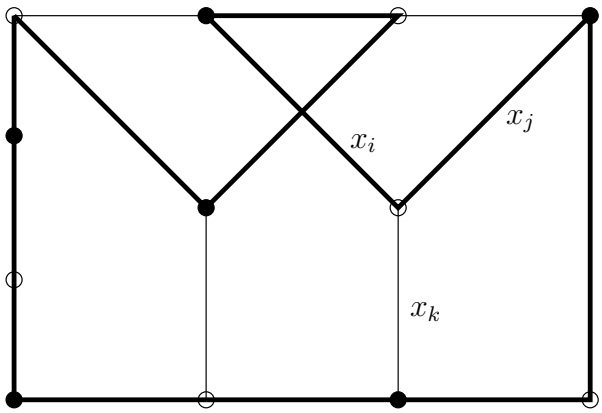

Figure 5.6: A perfect 2-matching with no cycles of length six and excluding the edge x_k.

For every variable gadget set the corresponding variable, x_i to true if the edge e_{true} is in the matching and otherwise to false. The xor gadget makes sure that the edges x_i in the clause gadgets are not in the matching. For every clause gadget there is no other 2-matchings then the ones shown in Figure 5.4, 5.5 and 5.6. Hence all clauses have exactly one true literal and we have a consistent assignment to the instance of One-In-Three 3SAT.

It is shown that there is a solution to B_6 if and only if there is a solution to the instance of One-In-Three 3SAT.

B_k

For any $k \geq 6$ the gadgets can be slightly modified to show that the problem is **NP**-complete as well. If k is even the problem B_k is similar to B_{k-1} since there is no cycles of odd length in a bipartite graph. Hence assume that k is even.

In the *variable gadget* add vertices until there are $k + 2$ vertices in the graph and the cycles are allowed in the matching.

The *xor gadget* need not to be modified since cycles of length six is not allowed for any $k \geq 6$ and the graph has the required property for any $k \geq 6$.

In the *clause gadget* let the left-most vertical line be divided by $k - 4$ vertices. In Figure 5.2 the line is divided by two vertices. Let the right-most vertical line be divided by at least $k - 10$ vertices. In Figure 5.2 the edge is not divided by any vertices. Then all cycles in Figure 5.4- 5.6 are allowed in the matching.

Chapter 6

Properties of the Cicular Betweenness Problem

Chor and Sudan write about betweenness when the points are on a line [6]. We consider a slightly modified problem when the points are on a circle and show that this version of the problem is **NP**-hard as well.

Definition 37 The Circular Betweenness problem: *A set U of n points, a set C of m orders with three points in each order. Find an permutation of the points on a circle such that as many orders as possible have their points in clock-wise order.*

When a point a is said to be *between* the points b and c on the circle we mean in clock-wise direction.

6.1 The Betweenness problem is NP-complete

The problem is obviously in **NP**.

Reduction from 3-SAT to Circular Betweenness

In 1998 Chor and Sudan showed that if the points are ordered on a line the problem is **NP**-complete. They reduce Max 3-SAT to that problem. In this section we use some of the ideas by Chor et. al when we reduce Max 3-SAT to Circular Betweenness.

Definition 38 The Maximum 3-SAT problem: *A set U of n variables, a set C of m clauses with three literals, where a literal is a variable or a negated variable. Find a true assignment of the variables such that as many clauses as possible are satisfied by the assignment.*

Suppose we have an instance of Max 3-SAT with n variables and m clauses. Define two points T and F on the circle. For every variable x_i construct a point

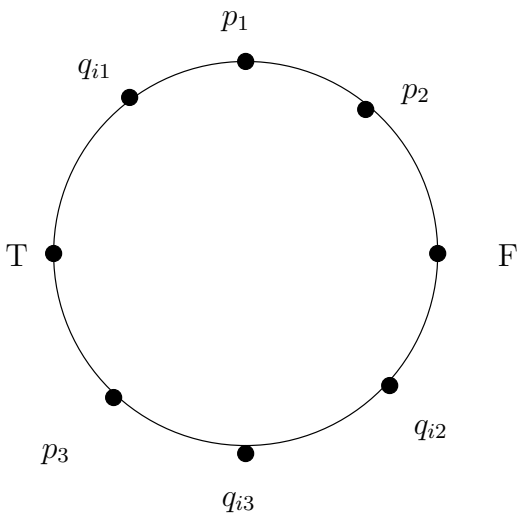

Figure 6.1: A clause $c_i = (x_1, \overline{x}_2, x_3)$ with the assignment x_1 and x_2 are true and x_3 is false represented as points on the circle.

p_i and for every clause $c_i = \{x_a, x_b, x_c\}$ construct three variable points q_{ia}, q_{ib}, q_{ic} and six extra points $r_{i1}, r_{i2}, r_{i3}, s_{i1}, s_{i2}, s_{i3}$.

If a variables x_i is true we want the point p_i to be between T and F and if x_i is false we want p_i to be between F and T (Figure 6.1). To give a variable a consistent value in every clause we construct for clause c_i two orders (T, q_{ia}, p_a) and (F, q_{ia}, p_a) (or (T, q_{ia}, p_a) and (F, p_a, q_{ia})if the variable x_a is negated). For every clause $c_i = \{x_a, x_b, x_c\}$ construct also the following orders: (q_a, q_b, q_c), (q_a, r_1, q_b), (q_b, r_2, q_c), (q_c, r_3, q_a), (r_1, s_1, r_2), (r_2, s_2, r_3), (r_3, s_3, r_1), (T, s_1, F), (T, s_2, F) and (T, s_3, F). For every variable there is one point on the circle and for every clause there are nine points and 16 orders. In total there is $2 + n + 9m$ points and $16m$ orders.

Lemma 37 *If there is a satisfying assignment to Max 3-SAT. Then there is a satisfying assignment to the constructed instance of Circular Betweenness.*

Proof 37 *If a variable x_i is true put the point p_i between T and F on the circle and if the variable x_i is false put the point p_i between F and T. For every clause $c_i = \{x_a, x_b, \overline{x}_c\}$, if a variable x_a is not negated put q_{ia} on the same side of T and F as p_a and if the variable x_a is negated put q_{ia} on the opposite side of T and F as p_a. Between F and T and between T and F put first all points of the type q_{ia} and then all points of the type p_a. The points then obey the orders of the form (T, q_{ia}, p_a). Permute the points q_{ia}, q_{ib} and q_{ic} such that they obey the order (q_{ia}, q_{ib}, q_{ic}), this can be done since at least two q's are on the same side of F and T and only occur*

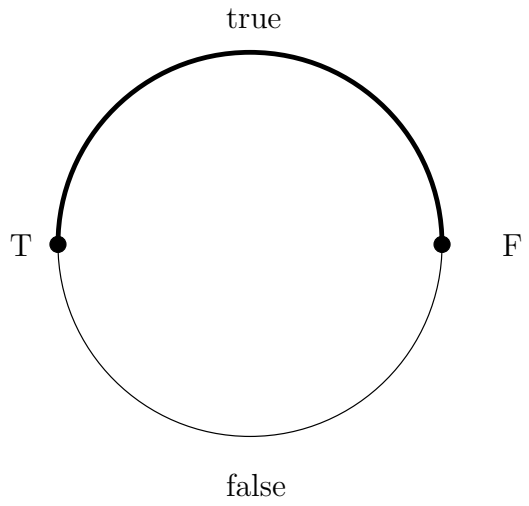

Figure 6.2: The true and false side of the circle.

in one order. Since the clause c_i is satisfied at least one variable x_a is true and by the construction one of the points q_{ia} is on the true side of the circle. For the remaining orders; put the points r_{ij} and s_{ij} on the true side if it is possible. Since at least one point q_{ia} is on the true side two points r_{ik} and r_{il} is on the true side. With two r_i-variables on the true side all three s_i-variables can be on the true side and all orders of the form (T, s_i, F) is satisfied. Hence all orders are satisfied.

Lemma 38 *Suppose there is an satisfying order of the points in the constructed instance of Circular Betweenness. Then there is an satisfying instance of Max 3-SAT.*

Proof 38 *If a point p_i is one the true side of the circle assign the variables x_i to true and false otherwise. Look at a clause c_i. Since all orders of the type (T, p, q) is satisfied the position of the q_{ji}-variables are consistent with the variable x_i. Suppose no variable q_i is on the true side of the circle. Then at most one r-variable is on the true side and at most two s-variables is on the true side. Then one of the orders (T, s_i, F) is not satisfied and we have a contradiction. Hence at least one of the variables q_{ij} is on the true side and the clause is satisfied.*

Lemma 39 *If there is no satisfying assignment to Max 3-SAT. Then for every not satisfied clause at most 15 of the 16 constructed orders are satisfied.*

Proof 39 *All clauses can not be true by Lemma 37 and 38. Suppose the clause c_i is not satisfied. By Lemma 38 all clauses can not be true and at least one of the orders is not satisfied. For example put one q_{ij} on the true side of the circle. Then*

one order of the type (T, p_j, q_{ij}) is false but all the rest can be true and 15 orders are satisfied.

Lemma 40 *A satisfiable instance of Max 3-SAT with m clauses is* **NP***-hard to approximate within $\frac{7}{8}m$ [17].*

Lemma 41 *For an satisfiable instance of Circular Betweenness with m orders it is* **NP***-hard to approximate within $\frac{127}{128}m$.*

Proof 40 *By Lemma 40 it is* **NP***-hard to approximate more than $\frac{7}{8}m$ clauses. From every clause we construct 16 orders and can satisfy all if the clause is satisfiable and at most 15 if the clause is not satisfiable. Hence we can satisfy at most a fraction of $\frac{\frac{7}{8}16m+\frac{1}{8}15m}{16m} = \frac{127}{128}$.*

6.2 Approximation algorithms

Put the points arbitrary on a circle. The ordering is said to be in clock-wise direction. If less than half of the orders is satisfied put the points in counter-clock-wise direction. Since every order is satisfied for one of the orderings at least half of the orders are satisfied. Hence the algorithm is in linear time and gives a 1/2-approximation.

6.3 Find permutation

Definition 39 *For a fix permutation of points on the circle and a black-box which gives an order of three points. How many orders do we need to decide the permutation?*

Theorem 11 *At least $O(n \log n)$ orders is needed.*

Proof 41 *There are $n!$ possible permutations. Each order eliminate half of them and we need at least $n \log n$ orders.*

Chapter 7

Conclusions

In Chapter 2 we construct two families, $\mathcal{H}$ and $\mathcal{G}$, of worst case graphs for the approximation algorithm by Frieze et al. for asymmetric TSP. With deterministic assumptions on the algorithm, it shows a worst case performance for graphs in the family $\mathcal{H}$. The algorithm returns for a graph $G_n \in \mathcal{H}$ by Theorem 1 a TSP tour of weight greater than $(opt(G_n) \cdot \log_2 n)/(2 + o(1))$. The analysis of the algorithm by Frieze et al. shows that the algorithm gives a TSP tour with weight less than or equal to $opt(G_n) \cdot \log_2 n$, hence the analysis of the algorithm by Frieze et al. is tight up to a factor of $1/2$. The ratio α between edges in different directions is greater than $n/2$ and the data dependent approximation algorithm by Frieze et al. is then proven to give an approximation better than $3\alpha/2 = 3n/4$ or $O(n)$.

With random assumption on the algorithm, it shows a worst case performance for the family $\mathcal{G}$ of symmetric graphs. The algorithm returns by Theorem 2 a TSP tour with expected weight

$$\frac{opt(G_n) - 1}{768}(\log_2 n - 5) - 1$$

What kind of algorithm might give a better approximation of the TSP tour on the worst case graphs? The reason that the spanning cactus in the algorithm by Frieze et al. is heavy is that many cycle covers are produced and that the cycles are connected simultaneously. One efficient heuristic in practice is to build *one* cycle cover and then patch the cycles together [18]. It is difficult to find a smart way of connecting the cycles and it is easy to construct graphs where the algorithm preforms poorly. In the algorithm by Frieze et al. the cycles connects few nodes (at most three) during the first $\log_3 m$ iterations. Larger cycles or some other subgraph would be useful. However finding the minimum cycle cover with large cycles is difficult since finding a minimum cycle cover with cycles of length at least three is **NP**-hard [32].

The new algorithm by Bläser is a development of the algorithm by Frieze et al. and uses *partial* cycle covers. In their recent algorithm Kaplan et al. [21] introduce

some new ideas. Basically, they compute a fractional cycle cover with certain 2-cycle constraints and then use such covers to extract cycle covers with few 2-cycles for the underlying graph. An interesting question is to investigate if the family of graphs defined in this paper—or any other class of graphs—give an approximation ratio $\Omega(\log n)$ for these new algorithms.

The question by Karp [23] as to whether there is a polynomial time heuristic for which the approximation ratio of asymmetric TSP is bounded by a constant is still open.

In order to construct an algorithm for asymmetric TSP it is natural to try to generalise the ideas used by Christofides [7]. In particular, it seems fruitful to search for structures similar to that of a spanning tree in asymmetric graphs. One such structure is the spanning cactus. A spanning cactus is also produced by the algorithm by Frieze et al.. In Chapter 3 we observe however, that finding the minimum spanning cactus and the minimum TSP tour in an asymmetric weighted complete graph are polynomial time equivalent problems. They also have the same hardness of approximation. Therefore it can not be easier to find a minimum spanning cactus than a minimum TSP tour. By reducing One-In-Three 3SAT we also show that determining if a general graph has a spanning cactus is **NP**-complete. An interesting field for using spanning cacti is for TSP with multiple salesmen, especially when the underlying graph is not Hamiltonian.

Definition 40 k-Travelling Salesman Problem (k-TSP) *is the following: given a graph G and a start vertex v_0, decide if there are k subtours each containing v_0, such that every other vertex in the graph is in exactly one tour.*

A k-TSP tour is obviously a spanning cactus since it consists of k disjoint cycles which start in the same vertex. It remains to be seen if cacti can be used in an approximation algorithm for k-TSP.

An instance of the *Asymmetric TSP* can be transformed into an instance of the *Symmetric TSP* in a certain related bipartite graph. In Chapter 4 we show, that, in spite of the fact that the edge weights in this bipartite graph do not obey the triangle inequality, the weights of minimum TSP tours in the two corresponding instances are within constant factors of each other. The symmetric bipartite realization of instances of Asymmetric TSP therefore open up new possibilities to construct both stronger approximation algorithms for the problem as well as stronger approximation hardness results.

Theorem 12 *Let G be a complete, weighted, asymmetric graph and suppose that the weights satisfy the triangle inequality. Denote by $w(T_A)$ the weight of a minimal TSP tour in G and by $w(T_B)$ the weight of a minimal TSP tour in $B(G)$. Then it holds that $w(T_A) \leq w(T_B) \leq 3w(T_A)$, or, equivalently, that $\frac{1}{3}w(T_B) \leq w(T_A) \leq w(T_B)$.*

Apart from the results presented in this chapter, it is, of course, possible to formulate, and prove, several other lemmas that relate structures in directed graphs

to structures in the corresponding bipartite realizations. We have, however, been unable to completely characterise the kind of Eulerian subgraphs of directed graphs that correspond to Hamiltonian cycles in the bipartite realisation. It seems that some kind of "locality" property is needed here—the Eulerian subgraph constructed in the proof of Theorem 7 is very "local" in the sense that it consists of a TSP tour augmented with edges that make very short jumps along the direction of the tour.

An algorithmic application, or, for that matter, a proof of approximation hardness, probably requires some additional results that describe the structure of the bipartite realization of a complete graph where the edge weights obey the triangle inequality. We have not been able to find any simple relation between weights of different edges in the bipartite realization other than the direct implication of the triangle inequality: that $w(\{s_u, t_v\}) + w(\{s_v, t_w\}) \leq w(\{s_u, t_w\})$.

When approximating asymmetric TSP Frieze et al. use cycle covers. In Chapter 4 we investigate connections between asymmetric and bipartite graphs and are hence interested also in cycle covers in bipartite graphs. In a symmetric bipartite graph all cycles have even length and the shortest cycle has length four. It has been conjectured [15] that finding a 2-matching without cycles of length six or less in a bipartite graph is **NP**-hard but as far as we know no proof has been published. We give a proof by reducing One-In-Three 3SAT and hence close the gap for bipartite graphs.

Chor and Sudan write about betweenness when the points are on a line [6]. In Chapter 6 we consider a slightly modified problem when the points are on a circle and show that this version of the problem is **NP**-hard as well.

Bibliography

[1] David Applegate, Robert Bixby, Vasek Chvátal, and William Cook. Concorde home, 2006. Website, <http://ww.tsp.gatech.edu/concorde/index.html>.

[2] Norbert Ascheuer, Matteo Fischetti, and Martin Grötschel. A polyhedral study of the asymmetric traveling salesman problem with time windows. *Networks*, 36(2):69–79, 2000.

[3] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation. Combinatorial Optimization Problems and their Approximability Properties.* Springer-Verlag, Berlin, 1999.

[4] Egon Balas. The prize collecting traveling salesman problem and its applications. In Gutin and Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 663–696. Kluwer Academic Publishers, 2002.

[5] Markus Bläser. A new approximation algorithm for the asymmetric tsp with triangle inequality. In *Proceedings of 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 639–647. SIAM, 2003.

[6] Benny Chor and Madhu Sudan. A geometric approach to betweenness. *SIAM Journal on Discrete Mathematics*, 11(4):511–523, 1998.

[7] Nicos Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report CS-93-13, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, 1976.

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition.* the MIT Press, 2001.

[9] Gerard Cornue'jols and William Pulleyblank. A matching problem with side conditions. *Discrete Mathematics*, 29:135–159, 1980.

[10] Lars Engebretsen. *Approximate Constraint Satisfaction.* Högskoletryckeriet, Stockholm, 2000. Doctoral Thesis.

[11] Lisa Fleischer. Building chain and cactus representations of all minimum cuts from hao-orlin in the same asymptotic run time. *Journal of Algorithms*, pages 51–72, 1999.

[12] Alan M. Frieze, Giulia Galbiati, and Francesco Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks*, 12:23–39, 1982.

[13] Michael R. Garey and David S. Johnson. *Computers and Intractability: a guide to the theory of NP-completeness*. W. H. Freeman and Company, San Fransisco, 1979.

[14] L. Gouveia and S. Voß. A classification of formulations for the (time dependent) traveling salesman problem. *European Journal of Operationnal Research*, 83:69–82, 1995.

[15] David Hartvigsen. Finding maximum square-free 2-matchings in bipartite graphs. *Journal of Combinatorial Theory Series B*. To appear.

[16] David Hartvigsen. The square-free 2-factor problem in bipartite graphs. In *Proceedings of Integer Programming and Combinatorial Optimization*, pages 234–241. Springer-Verlag, 1999.

[17] Johan Håstad. Some optimal inapproximability results. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, pages 1–10, El Paso, Texas, May 1997. ACM.

[18] David S. Johnson, Gregory Gutin, Lyle A. McGeoch, Anders Yeo, Weixiong Zhang, and Alexei Zverovitch. Experimental analysis of heuristics for the atsp. In Gutin and Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 445–487. Kluwer Academic Publishers, 2002.

[19] David S. Johnson and Lyle A. McGeoch. Experimental analysis of heuristics for the stsp. In Gutin and Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 369–443. Kluwer Academic Publishers, 2002.

[20] David S. Johnson and Christos H. Papadimitriou. Computational complexity. In Eugene L. Lawler, Jan K. Lenstra, Alexander H. G. Rinnoy Kan, and David B. Shmoys, editors, *The Traveling Salesman Problem*, chapter 3, pages 37–85. John Wiley & Sons, New York, 1985.

[21] Haim Kaplan, Moshe Lewenstein, Nira Shafrir, and Maxim Sviridenko. Approximation algorithms for asymmetric tsp by decomposing directed regular multigraphs. In *Proceedings of 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 56–65. IEEE Computer Society, 2003.

[22] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[23] Richard M. Karp. The fast approximate solution of hard combinatorial problems. In *Proceedings 6th South Eastern Conference on Combinatorics, Graph Theory and Computing*, pages 15–21. Utilitas Mathematica, Winnipeg, 1975.

[24] S. R. Kosaraju, J. K. Park, and C. Stein. Long tours and short superstrings. In *Proceedings of 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 166–177. IEEE Computer Society, 1994.

[25] Anna Palbom. Worst case preformance of an approximation algorithm for asymmetric tsp. In *Proceedings of 21ST Annual Symposium on Theoretical Aspects of Computer Science*, pages 465–476. Springer-Verlag, 2004.

[26] Anna Palbom. Complexity of the directed spanning cactus problem. *Discrete Applied Mathematics*, 146:81–91, 2005.

[27] Christos H. Papadimitriou and Santosh Vempala. On the approximability of the traveling salesman problem. Manuscript, `<http://www.cs.berkeley.edu/~christos/tsp.ps>`, 2002.

[28] Abraham P. Punnen. The traveling salesman problem: Applications, formulations and variations. In Gutin and Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 1–28. Kluwer Academic Publishers, 2002.

[29] Shmuel Safra and Oded Schwartz. On the complexity of approximating tsp with neighborhoods and related problems. *Computional Complexity*, 14(4):281–307, 2005.

[30] Sartaj K. Sahni and Teofilo Gonzalez. P-complete approximation problems. *Journal of the Assoc. Comput. Mach*, 23, 3:555–565, 1976.

[31] Günter Schaar. Remarks on hamiltonian properties of powers of digraphs. *Discrete Applied Mathematics*, 51:181–186, 1994.

[32] Leslie G. Vailiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.