Applications of Circulations and Removable Pairings to TSP and 2ECSS

Yao Fu

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for a master degree in

COMPUTER SCIENCE

Ottawa-Carleton Institute for Computer Science
University of Ottawa

Abstract

In this thesis we focus on two NP-hard and intensively studied problems: The travelling salesman problem (TSP), which aims to find a minimum cost tour that visits every node exactly once in a complete weighted graph, and the 2-edge-connected spanning subgraph problem (2ECSS), which aims to find a minimum size 2-edge-connected spanning subgraph in a given graph.

TSP and 2ECSS have many real world applications. However, both problems are NP-hard which means it is unlikely that polynomial time algorithms exist to solve them, so methods that return close to optimal solutions are sought. In this thesis we mainly focus on k-approximation algorithms for the two problems, which efficiently return a solution within k times of the optimal solution.

For a special case of TSP called graph TSP, using ideas from Mömke and Svensson, we present a $\frac{25}{18}$-approximation algorithm for a special class of graphs using circulations and T-joins, which improves the previous known best bound of $\frac{7}{5}$ for such graphs. Moreover, if the graph does not contain special nodes, our algorithm ensures the ratio of $\frac{4}{3}$.

For 2ECSS, given any k-edge-connected graph $G = (V, E),\ |V| = n,\ |E| = m$, we present an approximation algorithm that gives a 2-edge-connected spanning subgraph with the number of edges at most $n + \frac{m-n}{k-1} - \frac{k-2}{k-1}$ with a novel use of circulations, which improves both the approximation ratio and the simplicity of the proof compared to

a result by Huh in 2004.

Acknowledgments

Foremost, I would like to express my deep gratitude to my supervisor, Dr. Sylvia Boyd, whom provide me with every bit of guidance, assistance and expertise in combinatorial area, and giving me valuable feedback, advice and encouragement in our weekly meeting. It is a great pleasure to know her and work with her. I could not have imagined a better advisor and mentor for my master study .

I would like to thank the professors who guide me over the years, especially Caizhen Shan, Jianlong You, Quan Chen, Dr. Lucia Moura, Dr. Paola Flocchini and Dr. Amiya Nayak. I would also like to thank all my friends and labmates, in particular to Hong Pan, Yu Sun, Patrick Niesink, Dr. Maryam Haghighi, Tracy Ng, Ting Huyan, Maria Urlea, for their help, advice and suggestions.

I am grateful for the financial support from the University of Ottawa and OGS scholarship from the Province of Ontraio.

Last but not the least important, I would like to give special thanks to my family, my mother, my father, my boyfriend and Jasmine Chen. Without their love, support and encouragement, I would never image this thesis could be done.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

A salesman starts from a city wants to travel to a set of cities exactly once and back to the original city. Assume there is a path between every two cities. The problem to minimize the distance he or she travels can be modelled as the following: Let a finite set of nodes V represent all the cities and a set of edges E represent all the possible paths between cities. Then given a non-negative cost function $c \in \mathbb{R}^E$, where c_{uv} denotes the distance between nodes u and v, the *travelling salesman problem* (TSP) is to minimize the total cost of a cycle which goes to each node exactly once in the complete graph $G = (V, E)$.

The TSP has been widely used in real world applications, like vehicle routing, computer wiring, job sequencing and so on [Gar85], but there is no known algorithm that can solve the TSP in polynomial time[1], and it is known to be NP-hard [Kar72], so we strongly suspect no such polynomial time algorithm exists.

It is well-known that optimal tours for the TSP correspond to optimal solutions of a certain integer linear program [HK71]. The TSP tour can be expressed as a 0-1

[1]We usually use *time complexity* to measure the performance of an algorithm. Time complexity is usually determined by the input size of the problem, and expressed using the big O notation.

vector $x \in \mathbb{R}^E$, and the integer linear programming formulation (ILP) is as follows:

$$\begin{aligned}
\text{minimize} \quad & cx && (1.1)\\
\text{subject to} \quad & x(\delta(v)) = 2 \text{ for all } v \in V, && (1.2)\\
& x(\delta(S)) \geq 2 \text{ for all } S \subset V,\ 3 \leq |S| \leq n-3, && (1.3)\\
& 0 \leq x_e \leq 1 \text{ for all } e \in E, && (1.4)\\
& x \text{ integer}, && (1.5)
\end{aligned}$$

where $\delta(v)$ denotes the set of edges incident with v and $\delta(S)$ denotes the set of edges that have one end in S and the other end in $V \setminus S$, and for any set $F \subseteq E$, $x(F)$ denotes $\sum_{e \in F} x_e$. Constraints (1.2), (1.3) and (1.5) are respectively called the *degree constraints*, *subtour elimination constraints* and *integer constraints* of the TSP. If we drop constraint (1.5), then we get its linear programming relaxation (LP), also denoted as the *subtour elimination problem* (SEP). Although there is an exponential number of subtour elimination constraints in the LP, SEP can be solved in polynomial time [GLS88]. There exist many polynomial approaches to get a hopefully good solution to the TSP, but in this thesis we will mainly focus on a method called *approximation* which ensures a solution guaranteed to be close[2] to the optimal solution.

Since the TSP defined above is NP-hard, another question is whether we can solve the problem in polynomial time if we restrict the cost function. If the cost function c satisfies the triangle inequality:

$$c_{uv} + c_{vw} \geq c_{uw} \text{ for all } u, v, w \in V,$$

then we say such a cost function is *metric*, and we call the TSP with metric costs the *metric TSP*. Moreover, if we obtain the costs by starting with some graph $G' =$

[2]By "close" here, we mean that the solution returned by the approximation algorithm is bounded by some ratio. Please see Section 2.8 for more details.

(V', E'), letting all the edges in G' have cost 1, then add all edges uv which are not in the original graph, and take the cost of the minimum cost path from u to v as the cost of edge uv, then we say this problem is an instance of *graph TSP* for graph G'. It is easy to see that graph TSP is a special case of metric TSP. Unfortunately, metric TSP is still NP-hard [SLKL85], as is graph TSP [GKP95].

To study this problem from a polyhedral approach, we have to learn the polytopes[3] for the related problems. A point in a polytope is usually represented by a vector. The TSP polytope, which we denote by P_{TSP}, is the convex hull of the incidence vectors of TSP solutions. The polytope of the subtour elimination problem, P_{SEP}, consists of all $x \in \mathbb{R}^E$ that satisfy the constraints (1.2) to (1.4). From the two polytopes, we can see that if a vector is in the TSP polytope, then it also satisfies the constraints in the SEP polytope. Thus we know P_{TSP} is contained in P_{SEP}.

Since there are many ways to solve SEP, to measure the performance of the algorithm, we study the *integrality gap*[4] α for SEP, i.e. the worst case ratio between the optimal solutions of TSP and SEP. It is known that the integrality gap α is at most $\frac{3}{2}$ [SW90] in the metric case, but no example shows the ratio $\frac{3}{2}$ is tight. In fact, there is a well-known conjecture, open for over 30 years, as follows:

Conjecture 1.1.1. *The integrality gap α is at most $\frac{4}{3}$ for metric TSP.*

There is a family of graph TSP problems that show that if Conjecture 1.1.1 is true, it is the best possible. Denote the optimal solution for TSP and SEP by OPT_{TSP} and OPT_{SEP}. Consider the graph TSP problem for the graph shown in Figure 1.1(a).

[3]Please refer to Section 2.3 for definitions.
[4]Please refer to Section 2.7 for details.

The number of nodes in each row is k. We can see an optimal solution x for SEP is[5]:

$$x_e = \begin{cases} \frac{1}{2} \text{ where } e \in \{ab, bc, ca, de, ef, fd\}, \\ 1 \text{ where } c_e = 1 \text{ and not set to } \frac{1}{2}, \\ 0 \text{ otherwise .} \end{cases}$$

(See Figure 1.1(b)).

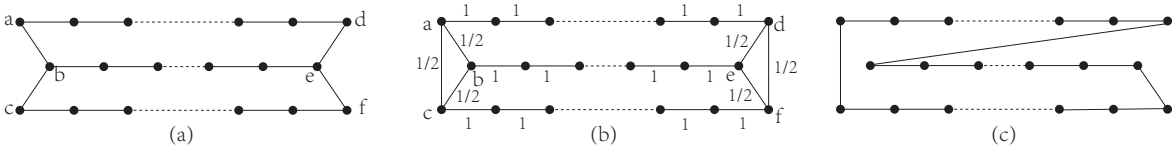


FIGURE 1.1: The graph that gives integrality gap of $\frac{4}{3}$.

Then the solution of the SEP has cost $\frac{1}{2} * 1 * 6 + 3(k-1) = 3k$. The optimal solution of the TSP is shown in Figure 1.1(c), the optimal tour has cost $2+1+3(k-1)+k = 4k$. This gives the ratio:

$$\frac{OPT_{TSP}}{OPT_{SEP}} = \lim_{k \to \infty} \frac{4k}{3k} = \frac{4}{3}. \tag{1.6}$$

There is a conjecture that says the worst case upper bound for the integrality gap for metric TSP happens when the optimal solution for SEP equals n [SWvZ12], so there has been some focus on trying to prove Conjecture 1.1.1 in this special case. One approach would be to prove it is true for every type of vertex of the SEP polytope. For graph TSP, this has been done for vertices with all components of value 1 or $\frac{1}{2}$ [MS11]. However, TSP for general graphs is hard to solve, so we focus on special graphs that satisfies the above condition. Then the support graph of a vertex[6] is a good choice for us.

[5]It can be easily seen that this solution is a feasible solution satisfying all the constraints for SEP, and the objective value is n. From Section 2.7, we know the lower bound of SEP in this case is n (n=3k). This solution reaches the lower bound, thus it is optimal.

[6]Please see Section 2.3 for definitions.

Another problem closely related to graph TSP is called the *2-edge-connected spanning subgraph* (2ECSS)[7] problem. The problem is to find a 2-edge-connected spanning subgraph of a given graph while minimizing the number of edges in the subgraph. Note that in the solution of 2ECSS, we can only use every edge at most once.

The ILP of 2ECSS can be obtained from the ILP of the TSP by relaxing the degree constraints:

$$\begin{aligned} \text{minimize} \quad & cx && (1.7)\\ \text{subject to:} \quad & x(\delta(S)) \geq 2 \text{ for all } S \subset V,\ 1 \leq |S| \leq n-1, && (1.8)\\ & 0 \leq x_e \leq 1, && (1.9)\\ & x \text{ integer.} && (1.10) \end{aligned}$$

2ECSS is widely used in network design. It is sometimes very important that the whole network stays connected after the loss of one link. To securely connect a network with the fewest links, the minimum cost 2-edge-connected spanning subgraph is naturally a good choice for such a problem. However, 2ECSS is NP-hard [JRV03] and also MAX SNP-hard even for cubic graphs[8] [CKK02].

1.2 Literature Review

For the travelling salesman problem:

In 1976, Christofides [Chr76] presented a simple and elegant proof which gives a $\frac{3}{2}$ approximation for metric TSP. Since then, no progress has been made for the general metric problem, but progress has been made for graph TSP. In 2011, Gharan et al. [GSS11] gave an approximation algorithm which improves Christofides' result

[7] 2-edge-connected mean the graph stays connected after the deletion of any edge, please see Section 2.1 for details.

[8] A cubic graph is a graph with degree three for every node.

to $\frac{3}{2} - \varepsilon$ on graph TSP. In the same year, Mömke and Svensson [MS11] presented a 1.461-approximation algorithm using the idea of circulations, T-joins[9] and removable pairings, which motivated many of the ideas of this thesis. Later, Mucha [Muc11] generalized their idea and provided a lower bound for the circulation, thus reducing the ratio to $\frac{13}{9}$. One year later, Sebő and Vygen [SV12] improved the approximation guarantee for graph TSP to $\frac{7}{5}$ by introducing the technique of ear decomposition combined with T-joins, and it is the best known result for graph TSP so far.

For special graphs, more and more results give the ratio of $\frac{4}{3}$. In 2005, Gamarnik et al. [GLS05] gave an approximation algorithm for cubic 3-edge-connected graphs which ensures the ratio is at most $\frac{3}{2} - \frac{5}{389}$. In 2011, Boyd et al. [BSvdSS11] presented a $\frac{4}{3}$-approximation algorithm for cubic 2-edge-connected graphs. In the same year, Mömke and Svensson's [MS11] algorithm gave $\frac{4}{3}$ on subcubic[10] 2-edge-connected graphs.

For the 2-edge-connected spanning subgraph problem:

In 1994, Khuller and Vishkin [KV94] gave a $\frac{3}{2}$-approximation algorithm. The ratio was improved later to $\frac{17}{12}$ by Cheriyan et al. in 1998 [CSS98] using ear decomposition. Two years later, the bound improved to $\frac{4}{3}$ by Vempala and Vetta [VV00]. In 2001, Krysta and Kumar [KK01] presented a $\frac{4}{3} - \varepsilon$ algorithm. In 2012, Sebő and Vygen [SV12] gave a $\frac{4}{3}$-approximation method with an easier and more elegant proof.

2ECSS still remains NP-hard even on cubic graphs. Krysta and Kumar [KK01] present a $\frac{21}{16} + \varepsilon$-approximation algorithm for cubic graphs. One year later, Csaba et al. [CKK02] showed they can find a $\frac{5}{4} + \varepsilon$ approximation algorithm for subcubic graphs. In 2004, Huh [Huh04] presented an algorithm guaranteeing $|V| + \frac{|E|-|V|}{k-1}$ for k-regular[11] graphs, moreover, if k=3, 4, 5 or 6, the approximation algorithm ensures

[9] Please refer to Section 4.1 for definitions.

[10] A subcubic graph is a graph with degree at most three for every node.

[11] A k-regular graph is a graph with degree k on every node.

$\frac{5}{4}, \frac{4}{3}, \frac{11}{8}$ or $\frac{7}{5}$. In 2003, Boyd et al. [BIT13] show the ratio is less than or equal to $\frac{6}{5}$ for cubic 3-edge-connected graphs.

1.3 Contributions and Outline

In this thesis, we study the use of circulations and T-joins for graph TSP and 2ECSS, mainly focusing on solving graph TSP on special graphs[12] and solving 2ECSS on k-edge-connected graphs. The main contributions are as follows:

1. For 2ECSS on k-edge-connected graphs, we present a simple approximation algorithm based on circulations that finds a 2-edge-connected spanning subgraph with the number of edges at most $n + \frac{m-n}{k-1} - \frac{k-2}{k-1}$. This improves Huh's [Huh04] result by the modest amount of $\frac{k-2}{k-1}$. However the main contribution here is that the proof provided is much simpler.

2. We prove some new results on T-joins. Similar types of results have been instrumental in the methods used in [MS11] and [SV12] for graph TSP, so we feel results such as the ones we provide could be useful in the future.

3. We present a simpler and clear explanation for some of the results stated in [MS11] for graph TSP, by combining the ideas and results found in [SV12] and [BSvdSS11] with those in [MS11].

4. We attempt to prove Conjecture 1.1.1 is true for graph TSP in the special case where the optimal value is n and this optimal is obtained by a $\frac{1}{3}$-integer vertex. This special case has not previously been studied. We present an approximation algorithm that returns a TSP tour with at most $\frac{25}{18}n$ edges in the support graph of any $\frac{1}{3}$-integer vertex, which improves on the previous best known bound for such graphs of $\frac{7}{5}$. Moreover, if the number of nodes in the graph is less than or

[12] We are dealing with the support graphs of $\frac{1}{k}$-integer vertex. Please see Chapter 6 for details.

equal to 9 or the tree we construct lacks certain types of nodes which we define later, the algorithm ensures a ratio of $\frac{4}{3}$.

The structure of the rest of this thesis is as follows:

In Chapter 2, we give definitions and notations from Section 2.1 to Section 2.3. Then we introduce three basic structures that are used throughout the thesis. We also study the polynomial time algorithms that solve those problems. In the last two sections of this chapter, we study the integrality gap of TSP, and introduce two well known approximation methods for metric TSP.

In Chapter 3, we give some background for circulations by introducing the associated polytope and related theorems. Then we present the basic structure of an algorithm for 2ECSS using circulations. We apply it to k-edge-connected graphs in Section 3.4, and achieve a result better than Huh's [Huh04] result both in the approximation ratio and simplicity of the proof.

In Chapter 4, we study the two polytopes for T-joins, then prove the equivalence of these two polytopes under certain conditions. We also provide some special vectors that are contained in the T-join polytope, which we think might be useful in the future.

In Chapter 5, we combine the results in [MS11], [SV12] and [BSvdSS11] using circulations and T-joins to present some of the important results in [MS11] in a simpler, clearer way. In the last two sections, we focus on two known methods to get removable pairs: By depth first search tree and by ear decomposition.

In Chapter 6, we present an approximation algorithm combining circulations and removable pairings and based on the method in [MS11] to get a TSP tour from the support graph of the $\frac{1}{3}$-integer vertex.

Finally, in Chapter 7, we give the conclusion, and describe some possible ways we can focus on continuing the research in the future.

Chapter 2

Background and Preliminaries

This chapter gives an introduction to basic definitions and theorems in graph theory, linear programming and polyhedra theory, then introduces some basic structures like depth first search trees, perfect matchings and ear decompositions which are useful for the chapters which follow. In the last two sections, we discuss the usefulness of integrality gap and two basic approximation algorithms for metric TSP.

2.1 Graph Theory

A *graph* $G = (V(G), E(G))$ consists of two parts, *nodes* and *edges*, where $V(G)$ denotes the set of nodes and $E(G)$ denotes the set of edges with every e in $E(G)$ joining two nodes in $V(G)$. For simplicity, we may use V and E to denote $V(G)$ and $E(G)$. The standard definitions for graph theory can be found in Bondy and Murty [BM08], and Schrijver [Sch03]. According to whether the edges have directions or not, a graph can be classified as a *directed graph* or *undirected graph*.

Given an *undirected graph* $G = (V_G, E)$ and *directed graph* $D = (V_D, A)$, then for an edge $e_1 = u_1v_1 \in E$ in G, we call u_1 and v_1 the two *ends* of edge e_1, we say u_1 and v_1 are *adjacent*, and both of them are *incident* with edge e_1. Then for an edge $e_2 = (u_2, v_2) \in A$ in D, we call e_2 a *directed edge* directed from u_2 to v_2, we say u_2 is

the *tail* and v_2 is the *head* of edge e_2, also, e_2 is an incoming edge directed into v_2 and an outgoing edge directed out from u_2. For any node $v_1 \in V_G$, we call the number of edges incident with v_1 the *degree* of v_1, denoted $deg(v_1)$ or d_{v_1}. For any node $v_2 \in V_D$, the number of edges coming into v_2 is the *indegree* of v_2, denoted $deg^{in}(v_2)$, the number of edges directed out is the *outdegree* of v_2, denoted $deg^{out}(v_2)$.

If every edge $e \in E$ in graph G is associated with a real number w_e, then we call w_e is the *weight* or *cost* of edge e, and say graph G is a *weighted graph*. The weights for a graph G can be written as a vector $w \in \mathbb{R}^E$. This can also be applied to a directed graph.

In an undirected graph G, if there exists edges $e_1, e_2 \in E(G)$ with the same two ends, i.e. some edges have more than one copy, then we say e_1 and e_2 are *parallel*, and G is a *multigraph*. For a directed graph D, if there exists two or more edges with the same head and same tail, then we say D is a multigraph.

In this thesis, we will use the notation shown below. Without special notification, we use graph to mean an undirected graph.

Given a set S:

- we use $|S|$ to denote the cardinality or size of S.

Given a graph $G = (V, E)$:

- $V(G)$ denotes the set of all the nodes in graph G. Usually we use n to represent $|V(G)|$.
- $E(G)$ denotes the set of all the edges in graph G. Usually we use m to represent $|E(G)|$.
- $\delta(S)$: For a node set $S \subset V$, $\delta(S)$ denotes the set of all edges $uv \in E$ with $u \in S$ and $v \in (V(G) \setminus S)$. The edge set $F = \delta(S)$ is called a *cut*. If S only contains a node v, then $|\delta(S)|$ or $|\delta(v)|$ denotes the degree of node v.

- $\lambda(S)$: for a node set $S \subset V(G)$, $\lambda(S)$ denotes the set of all edges $uv \in E$ with $u \in S$ and $v \in S$. If $S = V$, then $\lambda(S) = E$.

A *path* P starting from node r and ending at node s in G is a finite sequence of distinct nodes $r = v_0, v_1, ..., v_k = s$, and the adjacent nodes $v_{i-1}v_i$ form an edge $e_{i-1} \in E$ for $i = 1, 2, ..., k$. If the path P traverses all the nodes exactly once in G, i.e. $V(P) = V$, we call it a *Hamiltonian path*. If every two nodes in G are connected by a path, then we say G is *connected*. If nodes in path P are not necessarily distinct, then we say P is a *walk*.

A *cycle* C is a walk where the start node is the same as the end node, and all the other nodes are distinct and not the same as the start node. If the cycle C only contains one node, then we say C is a *loop*. If a cycle C traverses all the nodes exactly once in G, then we say C is a *Hamiltonian cycle*. Moreover, if graph G contains a Hamiltonian cycle, then G is *Hamiltonian*. A *tour* in a graph G is a closed walk that traverses each edge of G at least once, an *Euler tour* is a tour traverse each edge exactly once. If graph G contains an Euler tour, then G is an *Eulerian graph*.

Theorem 2.1.1. ***Euler's Theorem***

A graph $G = (V, E)$ has an Euler tour if and only if G satisfies the following two constraints:

1. *G is connected.*

2. *For all $v \in V$, $deg(v)$ is even.*

To find an Euler tour in G in polynomial time, we can use Hierholzer's Algorithm.

Hierholzer's Algorithm [HC73]

Input: A connected graph $G = (V, E)$ with even degree everywhere.

Output: An Euler tour in G.

Step 1 : Pick any node r, set C^* to null and p, p^* to r.

Step 2 : Find a cycle C starting at p and ending at p, mark all nodes in C as visited, insert C into C^*, and set p^* to p.

Step 3 : Set the next unvisited node next to p^* in C^* to p, repeat Step 2 until we cannot find any available node.

Step 4 : Return C^*.

This algorithm can be implemented in $\Theta(m)$ time [Fle91].

A *tree* T is a connected graph which contains no cycles. A tree has three main properties:

1. Connected, but disconnected after deleting any edge;
2. Acyclic;
3. $|E(T)| = |V(T)| - 1$.

T is called a *rooted tree* if one node is designated as a *root*. All the other degree one nodes in T are the *leaf* nodes and the rest of the nodes are the *internal nodes*. In the tree, there is a unique path from the root to all the other nodes, and the length of the longest such path is the *height* of the tree.

A *forest* is a graph without cycles.

A graph $H = (V(H), E(H))$ is called a *subgraph* of G if $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$. Moreover, if $V(H) = V(G)$, then we say H is *spanning*. Given edge costs $c \in \mathbb{R}^{E(G)}$, a *minimum cost spanning tree* (MST) denotes the spanning tree with minimum cost, i.e. for any spanning tree ST with cost $c(ST)$, we have $c(MST) \leq c(ST)$. An MST can be computed in $\Theta(m + n \log n)$ time [TH01].

If $\deg(v) = k$ for all $v \in V(G)$, then we say G is a *k-regular* graph. If $k = 3$, then we can also say G is a *cubic graph*. If $\delta(v) \leq 3$ for all $v \in V(G)$, then G is a *subcubic* graph. If for any subset $S \subset V(G)$ we have $\delta(S) \geq k$, then we say G is a *k-edge-connected* graph. If any two nodes in V are adjacent, then we say G is

a *complete graph*, usually denoted by K_n, where n is the number of nodes in G. A graph G is *k-vertex-connected* if G stays connected after the deletion of $k-1$ nodes.

A *metric* on a set X is a vector $c = R^{X \times X}$, and for any $x,\ y,\ z \in X$, it satisfies the following conditions: $(1)c_{xy} \geq 0$; $(2)c_{xy} = c_{yx}$; $(3)c_{xy} + c_{yz} \geq c_{xz}$.

2.2 Linear Programming and Integer Linear Programming

A *linear programming problem*, abbreviated to LP, is a problem maximizing or minimizing a linear objective function subject to a set of linear equalities or inequalities. The standard form can be written as:

$$\begin{gathered} \text{maximize } \{\mathbf{cx} \mid \mathbf{Ax} \leq \mathbf{b};\ \mathbf{x} \geq 0\} \\ \text{or} \\ \text{minimize } \{\mathbf{cx} \mid \mathbf{Ax} \geq \mathbf{b};\ \mathbf{x} \geq 0\}, \end{gathered} \tag{2.1}$$

where $\mathbf{x}$ is a vector $(x_1,\ x_2\ , ..., x_n)$ representing the *decision variables*, $\mathbf{A}$ is an m x n matrix representing the coefficients, $\mathbf{b}$ is a vector of length m, and $\mathbf{c}$ is a vector of length n, $\mathbf{cx}$ is the *objective function*, and the rest of the equalities and inequalities are the *constraints*.

Given a vector x', if it satisfies all the constraints, then we say x' is a *feasible solution*, and it is *optimal* if it maximizes or minimizes the objective function.

If we add the *integer constraints* to (2.1), then we say this linear programming is

an *integer linear programming*, abbreviated to ILP, i.e.

$$\begin{gathered}\text{maximize } \{\mathbf{cx} \mid \mathbf{Ax} \leq \mathbf{b};\ \mathbf{x} \geq 0;\ \mathbf{x} \text{ integer}\} \\ \text{or} \\ \text{minimize } \{\mathbf{cx} \mid \mathbf{Ax} \geq \mathbf{b};\ \mathbf{x} \geq 0;\ \mathbf{x} \text{ integer}\}.\end{gathered} \tag{2.2}$$

If we add the binary constraint, i.e. restrict all variables to be one of the values in $\{0, 1\}$, then we call this ILP a *binary integer program*, abbreviated to BIP.

2.3 Polyhedral Theory

In the following section we will give some basic definitions and theorems in polyhedral theory, thus giving a more visual explanation for linear programming. For a good overview for polyhedral theory, please see Schrijver [Sch03] and Goemans [Goe07]. First we will introduce some definitions:

- **Half space:** A subset $H \subset \mathbb{R}^n$ is called an *affine halfspace* if $H = cx \leq \beta,\ c \in \mathbb{R}^n$ with $c \neq 0$ and $\beta \in \mathbb{R}$. If $\beta = 0$, then H is called a *linear halfspace*.

- **Polyhedron:** A *polyhedron* P is the intersection of a finite set of halfspaces, i.e. P=$\{x \in \mathbb{R}^n \mid Ax \leq \beta\}$ where A is a $m \times n$ matrix and $\beta \in \mathbb{R}^m$.

- **Face:** A *face* F in polyhedron $P = \{x \in \mathbb{R}^n \mid Ax \leq \beta\}$ is $F = \{x \in P \mid ax = b\}$ where $ax \leq b$ is a valid inequality for P. Note that the empty set or P itself are also considered faces. Equivalently, a face can be obtained by taking the whole linear system defining P and replacing some inequalities with equalities, or taking the empty set. We call F a *proper face* if it is not equal to P.

- **Dimension:** The *dimension* of a polyhedron P is $N - 1$ where N is the max-

imum number of affinely independent[1] points in P. P is *full dimension* if its dimension equals n. In other words, there does not exist a set of equations that are satisfied by all $x \in P$. The polytope of the travelling salesman problem is not full dimension because all points in the polytope satisfies the degree constraints.

- **Polytope:** A *polytope* is a bounded polyhedron.
- **Vertex or Extreme point:** A *vertex* or *extreme point* is a point $x \in P$ where x is a face of P of dimension zero. Note that a vertex can also be defined as the unique solution to a subset of linearly independent equations in $Ax = \beta$.
- **Support Graph:** Given a graph $G = (V, E)$, a polyhedron $P \subset \mathbb{R}^E$ and a vector $x \in P$, the *support graph* of x denotes the subgraph $G_x = (V, E_x)$ where E_x is the set of all edges $e \in E$ with $x_e > 0$.

Given the complete graph $G = (V, E)$, the constraints in (2.3) below define the *subtour polytope* P_{SEP} for the travelling salesman problem, and any points $x \in \mathbb{R}^E$ which satisfy the constraints are in this polytope.

$$\begin{aligned} &(i)\ x(\delta(v)) \geq 2 \text{ for all } v \in V, \\ &(ii)\ x(\delta(S)) \geq 2 \text{ for all } S \subset V,\ 3 \leq |S| \leq n-3, \\ &(iii)\ 0 \leq x_e \leq 1 \text{ for all } e \in E. \end{aligned} \tag{2.3}$$

There are some groups of points in the subtour polytope with special properties, for which we can use some techniques to get a"good"[2] TSP tour. Next we define some of these special points which we will use in the following chapters, let $x(F)$ denotes $\sum_{e \in F} x_e$ for any set $F \subseteq E$.

[1] A set of nodes $X \subseteq \mathbb{R}^n$ are affinely independent if $u_1 - u_0$, ..., $u_n - u_0$ are linearly independent.

[2] By "good" here we mean the final solution is near to the optimal TSP solution, more details can be seen in Section 2.7.

1. Special $\frac{1}{2}$-integer vertex: This denotes a vector x in the subtour polytope where all x_e values are $0,\frac{1}{2}$ or 1. In the support graph of x_e, all the $\frac{1}{2}$-edges form disjoint cycles and all the 1-edge form paths that connect the cycle nodes [3].

2. $\frac{1}{k}$-integer vertex: This denotes a vector x where all x_e values are of the form $\frac{i}{k}$ where $i \in [0, k]$. For example, for k=3, all x_e values are $0,\frac{1}{3},\frac{2}{3}$ or 1.

Example Let $G = (V, E)$ be the complete graph on 10 nodes with labels $A, B, ..., J$. Then the following x is a $\frac{1}{3}$-integer vertex of the subtour polytope:

$$x_e = \begin{cases} \frac{1}{3},\ e \text{ is } AB, DF, EG \text{ or } HJ, \\ \frac{2}{3},\ e \text{ is } AC, BC, CD, EF, HI, IJ \text{ or } GI, \\ 1,\ e \text{ is } AG, BE, FH \text{ or } DJ, \\ 0, otherwise \end{cases} \tag{2.4}$$

Figure 2.1 shows the support graph of this $\frac{1}{3}$ vector.

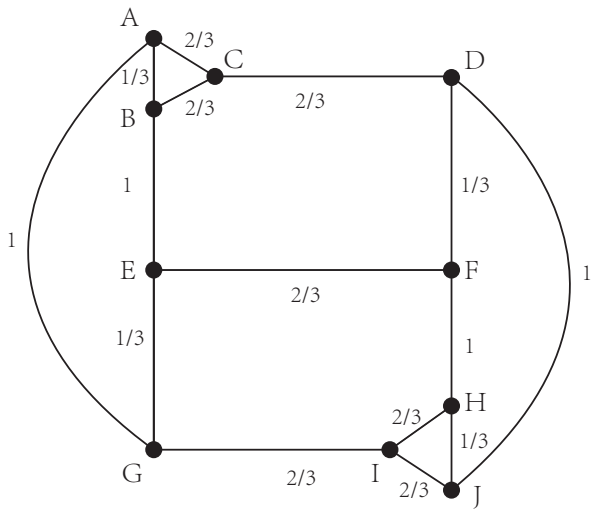


FIGURE 2.1: The support graph of the $\frac{1}{3}$-integer vertex x from (2.4).

[3]It is easily seen that the support graph of a special $\frac{1}{2}$-integer solution is cubic and 2-edge-connected.

Given a vector $x \in \mathbb{R}^n$, a set of points $x^{(1)}, x^{(2)}, ..., x^{(k)} \in \mathbb{R}^n$ and real numbers $\lambda_1, \lambda_2, ..., \lambda_k$ such that $0 \leq \lambda_i \leq 1$ for all $i = 1, 2, ..., k$, then x is said to be a *convex combination* of points $x^{(1)}, x^{(2)}, ..., x^{(k)}$ if

$$\begin{aligned} x &= \sum_{i=1}^{k} \lambda_i x^{(i)}, \\ \sum_{j=1}^{k} \lambda_j &= 1. \end{aligned} \tag{2.5}$$

Given a set of points $S \subset \mathbb{R}^n$, the *convex hull* of S is the set of all $x \in \mathbb{R}^n$ such that x is a convex combination of points in S. Note that given a polytope $P \subset \mathbb{R}^E$, P can also be defined as the convex hull of its vertices. A polytope where all extreme points are integer-valued is called an *integer polytope*. The following is a well-known theorem we will use in the following chapters.

Theorem 2.3.1. *Given an integer polytope $P \subset \mathbb{R}^n$ and a cost function $c \in \mathbb{R}^n$, then if there is a feasible solution x' in P, then there exist an integer solution x with $cx \leq cx'$.*

Proof. Since x' can be written as a convex combination of extreme points $x^{(i)}$ of P:

$$x' = \sum_{i=1}^{k} \lambda x^{(i)} \text{ where } \sum_{j=1}^{k} \lambda_j = 1,\ 0 \leq \lambda_j \leq 1 \text{ for } j = 1, 2, ..., k.$$

If we multiply both sides by c, then we have $cx' = \sum_{i=1}^{k} \lambda c x^{(i)}$. If for all $x^{(i)}$ we have $cx^{(i)} > cx'$, then we have

$$\begin{aligned} cx' &= \sum_{i=1}^{k} \lambda c x^{(i)} \\ &> \sum_{i=1}^{k} \lambda_i c x' \\ &= cx'. \end{aligned}$$

This leads to a contradiction, so there must exist some $x^{(i)}$ which satisfies $cx^{(i)} \leq cx'$. Because all extreme points are integer-valued, this proves there exists an integer solution x with $cx \leq cx'$.

So Theorem 2.3.1 is proved. □

Using similar ideas, we can also prove there exists an integer solution x with $cx \geq cx'$.

2.4 Depth First Search Tree

Breadth first search and depth first search are two main techniques that build special trees on a connected graph. For depth first search, we always keep on searching nodes as far as possible before backtracking. [Eve79].

The following algorithm describes how we get a depth first search tree in a graph.

Algorithm 2.1. DFS–Tree

Given a graph $G = (V, E)$, and an edge set T, set $T = \phi$.

Step 1 : Select an arbitrary node as the root and set $x \leftarrow r$;

Step 2 : Mark x as "visited", get the first unvisited neighbour n_x of x, if there is no such node, backtrack to the nearest unvisited node n_x.

Step 3 : If n_x is not empty, add edge (x, n_x) to set T, set $x \leftarrow n_x$, if all nodes are visited, return T. Otherwise, repeat Step 2.

At the end of the above algorithm, all the edges in set T form a depth first search tree. For example, consider the graph in Figure 2.2(a), which is the Peterson Graph. If we start from node F, then follow the path $F, A, E, J, G, I, D, C, B$, then we are stuck and cannot find any unvisited neighbours, so we backtrack to node C, then follow the edge CH, thus all these edges form a depth first search tree (see Figure 2.2(b)).

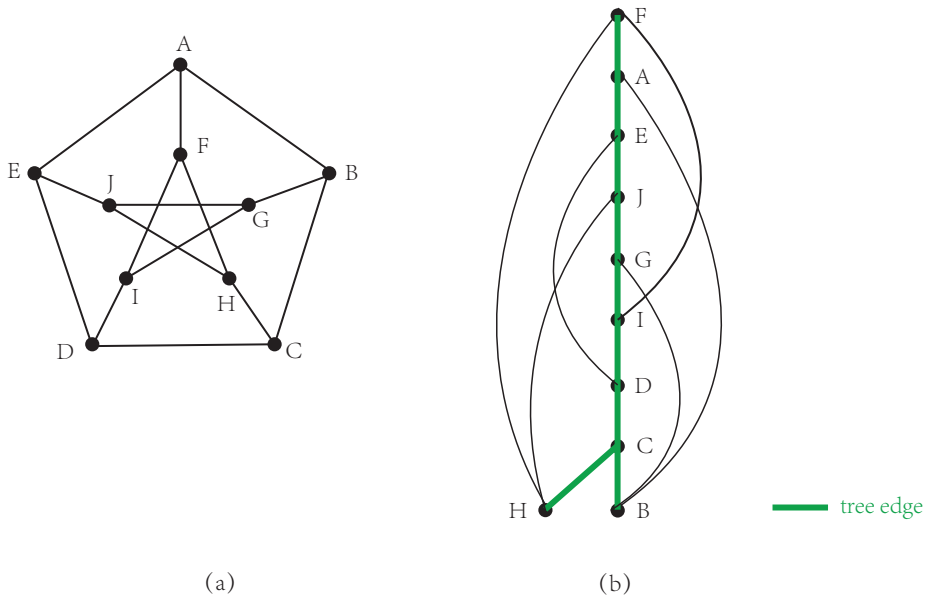


FIGURE 2.2: A depth first search tree for the Peterson Graph.

In the following chapters, we call the edges in a depth first search tree the *tree edges*, denoted by T, and the rest of the edges are called the *back edges*, denoted by B. If we delete any edge $e \in T$, the tree will decompose into two parts T_1 and T_2, and we define the *tree cut* for edge e to be the set of edges that have one end in $V(T_1)$ and the other end in $V(T_2)$.

2.5 Perfect Matching

Given a graph $G = (V, E)$ and an edge set $M \subset E$, M is called a *matching* if for all $v \in V(M)$, v has degree exactly one. For an edge $e \in M$ with two ends u, v, we say u, v are *saturated* by M. M is a *maximum matching* if G has no matching M' with $|M'| > |M|$. M is a *perfect matching* if $|V(M)| = |V|$.

For any graph $G = (V, E)$, the *perfect matching polytope* P_{PM} is the convex hull of all incidence vectors of perfect matchings of G. The following is a very powerful and useful theorem due to Edmonds [Edm65]:

Theorem 2.5.1. *The perfect matching polytope P_{PM} is the set of all $x \in \mathbb{R}^E$ satisfying the following [Edm65]:*

$$\begin{aligned} x_e &\geq 0 \text{ for all } e \in E, \\ x(\delta(v)) &= 1 \text{ for all } v \in V, \\ x(\delta(U)) &\geq 1 \text{ for all } U \subseteq V \text{ with } |U| \text{ odd }. \end{aligned} \tag{2.6}$$

If the graph G is bipartite, we only need the first two constraints in (2.6) to define P_{PM}. But for general graphs, if we only have the first two constraints, then for a triangle with three nodes, if we set all $x_e = \frac{1}{2}$, it satisfies the constraints, but we know we cannot find a perfect matching in a triangle, so we need to add the third set of constraints.

Figure 2.3 (a) shows a maximum matching M for the graph on the left-hand side, $M = \{AB, CD, EF, HI\}$. Note that we cannot find a perfect matching for 2.3(a). Figure 2.3 (b) shows a perfect matching for the Peterson graph.

Note that by Theorem 2.5.1, any feasible solution $x \in P_{PM}$ can be written as a convex combination of incidence vectors of perfect matchings.

Given a graph $G = (V, E)$ and edge costs $c \in \mathbb{R}^E$, a minimum cost perfect matching can be found in time $\Theta(n^2 m)$, and a maximum cost perfect macthing can also be found in time $\Theta(n^2 m)$ [Sch03].

Another structure similar to perfect matchings is called a *2-factor*. Given a graph $G = (V, E)$, a *2-factor* in G is a set of edges F such that $deg(v) \cap F = 2$ for all $v \in V$, i.e. the 2-factor decomposes the graph into a set of cycles that span V.

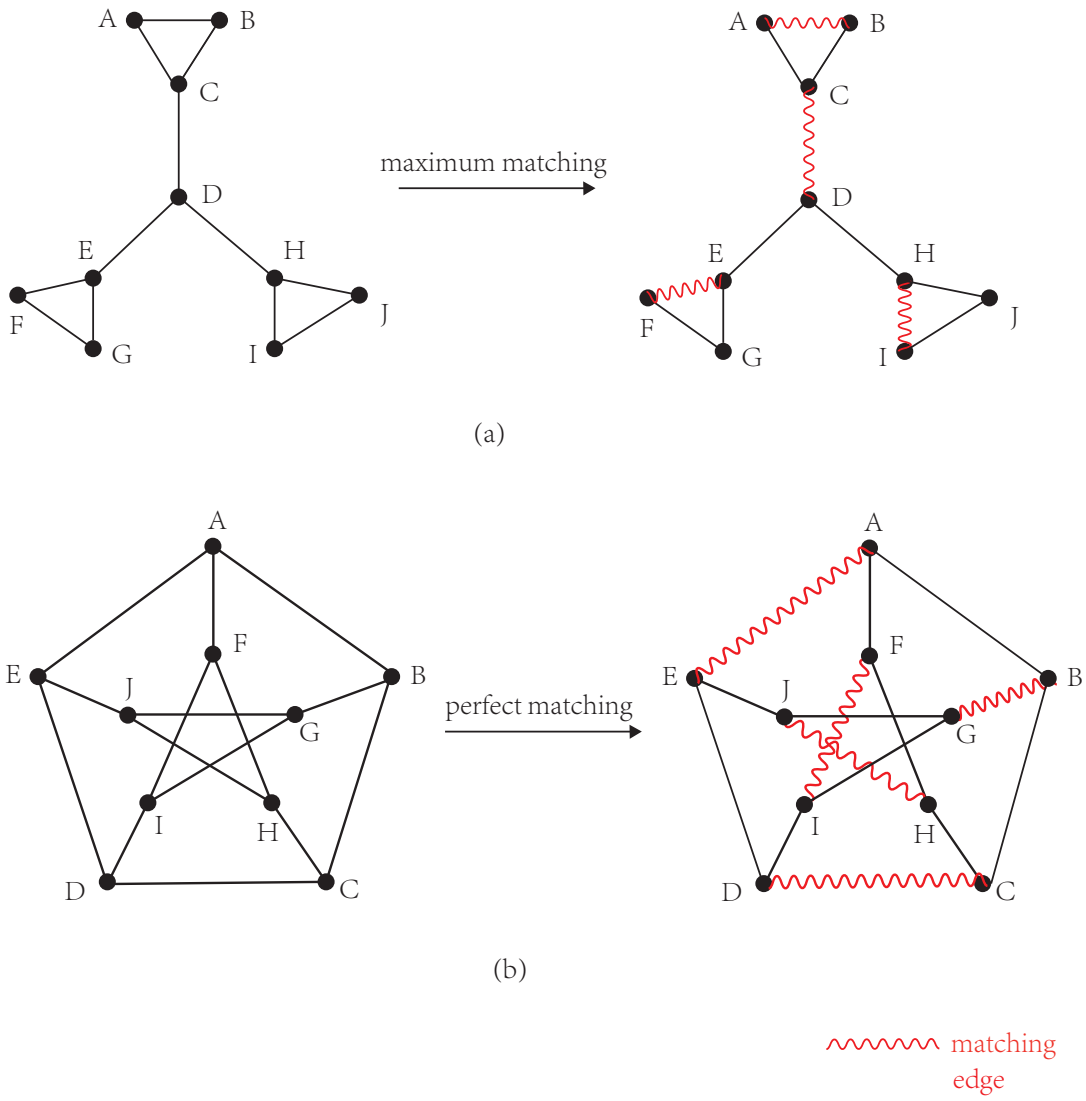


FIGURE 2.3: Graphs illustrating (a) a maximum matching, and (b) a perfect matching.

2.6 Ear Decomposition

An *ear decomposition* of an undirected graph $G = (V, E)$ is a partition of E into $P_0, P_1, ..., P_k$ where P_0 is a single node[4], and P_k is either:

1. A path which shares exactly two endpoints with $P_0 \cup P_1 \cup ... \cup P_{k-1}$, or
2. A cycle which shares exactly one endpoint with $P_0 \cup P_1 \cup ... \cup P_{k-1}$.

The sets $P_1, P_2, ..., P_k$ are called *ears*, and the nodes belong to an ear are the set of nodes inside this ear and do not include the endpoints. If the endpoints of P_i are inside P_j, then we say P_i is *attached* to P_j. An ear is *pendant* if there is no ear attached to it, and *non-pendant* otherwise. An ear with length l is called an $l-$ear, and if $l = 1$, then we call it a *trivial* ear. P_k is called an *open ear* if P_k is a path, and it is called a *closed ear* if P_k is a cycle. If all ears except P_1 are open ears, then we say this is an *open ear decomposition.*

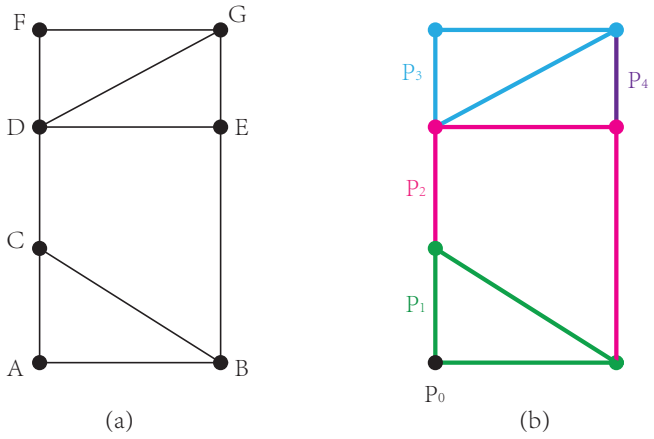


FIGURE 2.4: Graph illustrating an ear decomposition.

Example Given the graph G shown in Figure 2.4 (a), we can get the ear decomposition shown in Figure 2.4 (b) as follows:

- A is the start node, denote it as P_0.

[4]Sometimes P_0 is defined as a cycle [MR86]. In the following chapters, we refer to it as a single node.

- The first ear follows the cycle $ACBA$, denote it as P_1.
- The second ear follows the path $CDEB$, denote it as P_2. Note P_2 is an open ear and contains nodes D and E. The nodes C and B are the two endpoints of P_2.
- The third ear follows the cycle $DFGD$, denote it as P_3. P_3 is a closed ear attached to ear P_2.
- The last ear is a trivial ear and follows the path GE, denote it as P_4. No nodes belong to P_4.

Theorem 2.6.1. *[Has31] A graph G is a 2-edge-connected graph if and only if it has an ear decomposition. Moreover, G is a 2-vertex-connected graph if and only is it has an open ear decomposition.*

An ear decomposition can be computed in polynomial time. If we use a depth first search tree, it can be compute in $\Theta(\log^2(n))$ time with $\Theta(n^4)$ processors [Lov85].

2.7 Integrality Gap

For a BIP problem with all $x_e \in \{0, 1\}$, if we drop the integer constraint, i.e. relax the binary constraint to $0 \leq x_e \leq 1$, then the new linear programming problem is called a *relaxation* of the original 0-1 integer linear programming problem. For an ILP minimizing the objective function, the relaxation provides a lower bound for the ILP problem. In terms of theoretical computational complexity, there is a main difference between LP and ILP: ILP is usually NP-hard, while LP is polynomially-time solvable [Sch03]. So for a hard ILP problem for which it is highly impossible get the optimal solution in polynomial time, then we can solve its LP relaxation and get the corresponding lower (or upper) bound. After we get the solution for the LP relaxation, we can sometimes use other methods to transfer to a feasible solution to

the original integer linear programming problem. We can use the maximum ratio over all cost functions of the ILP optimal value and its LP relaxation optimal value to judge the quality of the bound provided by the LP relaxation, and we denote this ratio to be the *integrality gap* for the LP relaxation. If the ratio is small, that means we get a bound that is close to the optimal solution.

Consider the ILP of TSP defined in (1.1) to (1.5), if we also delete constraints (1.3) in its LP relaxation, we get a relaxation for the 2-factor problem, which is called the *fractional 2-factor problem*. We know the subtour elimination problem provides a lower bound for TSP, and the fractional 2-factor problem provides a lower bound for the subtour elimination problem. Then for the solution of any fractional 2-factor problem, we have

$$\begin{aligned} \sum_{v \in V} x(\delta(v)) &= 2|V|, \\ 2\sum_{e \in E} x_e &= 2|V| \\ \sum_{e \in E} x_e &= n. \end{aligned}$$

Then if all the edge costs are greater than or equal to 1, we know the lower bound for the fractional 2-factor problem is n, thus the lower bound for the subtour elimination problem is also n.[5]

2.8 Approximation Algorithms for TSP

There are various ways to get a feasible solution for TSP. The method we are going to use is called an approximation method. An algorithm is an *approximation algorithm* if it, in polynomially-time, returns a near optimal solution. It is called a *ρ-approximation algorithm* where ρ is a positive constant, if the solution from the ap-

[5]In the following context, we will deal with the graphs which the optimal value for subtour elimination problem is n. Please see Chapter 6 for more details.

proximation method SOL_{Approx} and the optimal solution for the original minimization problem OPT_{Min} satisfy the following inequality:

$$\frac{SOL_{Approx}}{OPT_{Min}} \leq \rho. \tag{2.7}$$

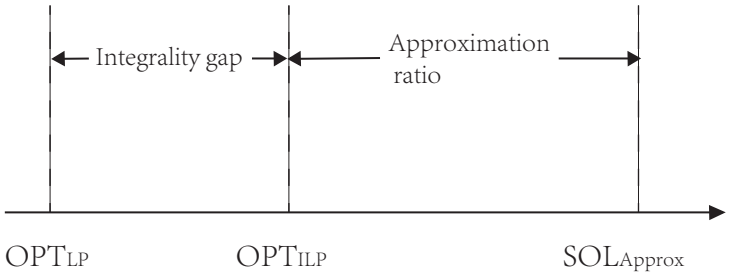


FIGURE 2.5: Graph which shows the relation between integrality gap and approximation ratio.

Figure 2.5 shows the relation between approximation ratio and integrality gap.

For the travelling salesman problem, there are two easy approximation algorithms: the 2-approximation method and Christofides' method. All these method follows the basic idea described below:

Basic Structure for Approximation Algorithm for metric TSP

Input: A complete graph $G = (V, E)$ with metric edge costs c.

Output: A TSP tour T_G within some ratio of the optimal solution.

Step 1 : Get a spanning tree T.

Step 2 : Adding edges to make the odd degree nodes in T to be even.

Step 3 : Get an Eulerian graph, then return the corresponding TSP tour.

Next we will introduce the two algorithms: 2-approximation method and Christofides' method ($\frac{3}{2}$-approximation method).

Algorithm 2.2. 2-approximation method

Input: A complete graph $G = (V, E)$ with metric edge costs c.

Output: A TSP tour with cost less than or equal to twice the optimal solution OPT_{TSP}.

Step 1 : Get a minimum cost spanning tree T in G, double all the edges in T to get a new graph G'.

Step 2 : Get the Euler tour $T_{G'}$ in G'.

Step 3 : Shortcut the Euler tour $T_{G'}$ to get a TSP tour T_G, and return T_G.

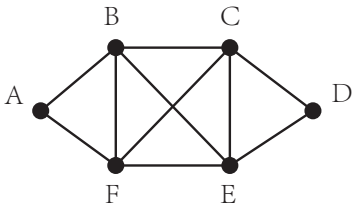
After Step 1, we know G' is a connected graph with even degree everywhere, thus from Theorem 2.1.1 we know there exists an Euler tour $T_{G'}$ in G'. Then we use the following theorem to show we can get a TSP tour with the desired approximation ratio.

Theorem 2.8.1. *Given a complete graph $G = (V, E)$ with metric edge costs $c \in \mathbb{R}^E$, if we can find an Euler tour $T_{G'}$ with cost $c(T_{G'})$ in G, then we can find a TSP tour T_G with cost $c(T_G) \leq c(T_{G'})$.*

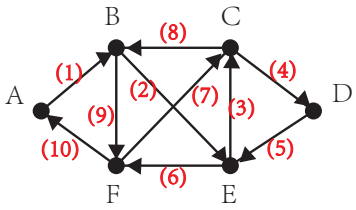
Proof. If we have an Euler tour $T_{G'}$, we can shortcut $T_{G'}$ to get a TSP tour T_G by deleting the nodes already visited in T_G (See Figure 2.6). Assume $T_{G'} = (v_0, v_1, ..., v_k)$, then suppose for node v_i, we have to visit it more than once, so the sequence near the second copy $v_y v_x v_z$ becomes $v_y v_x$. Since the costs are metric, so $c(v_y v_x) + c(v_x v_z) \geq c(v_y v_z)$, thus we know $c(T_G) \leq c(T_{G'})$. Hence Theorem 2.8.1 is proved. □

From Theorem 2.8.1 we know the cost of a TSP tour is less than or equal to the cost of an Euler tour, and the optimal solution of TSP must be greater than or equal to the cost of a minimum cost spanning tree, so we have the following:

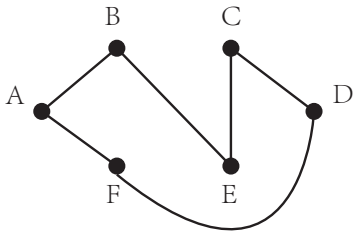
$$\begin{aligned} c(T_G) &\leq c(T_{G'}) \\ &= 2c(T) \leq 2OPT_{TSP}. \end{aligned}$$



(a) The graph shown above satisfies the conditions of Euler's theorem.



(b) Get an Eulerian tour as follows: "ABECDEFCBFA" Shortcut the tour by deleting the nodes that appeared before, i.e. delete the red nodes.



(c) The TSP tour obtained from the Euler tour : "ABECDFA" .

FIGURE 2.6: Graph showing how we transfer from an Euler tour to a TSP tour.

Christofides [Chr76] gave a $\frac{3}{2}$-approximation algorithm for metric TSP in 1976, and no result has been able to beat it ever since. Compared to the 2-approximation method, instead of doubling the whole tree, Christofides' algorithm only deals with the odd degree nodes, and finds a minimum cost perfect matching to fix the parity. The algorithm is as follows:

Algorithm 2.3. $\frac{3}{2}$-approximation method

Input: A complete graph $G = (V, E)$ with metric edge cost $c \in \mathbb{R}^E$.

Output: A TSP tour with cost less than or equal to $\frac{3}{2}$ the optimal solution $\mathbf{OPT_{TSP}}$

Step 1 : Get a minimum cost spanning tree T in G.

Step 2 : Find a minimum cost perfect matching M^* for all the odd degree nodes in T.

Step 3 : Get a Eulerian graph G_E by taking $T \cup M^*$, then use method described in Theorem 2.8.1 to get a TSP tour T_G, and return T_G.

Given any graph G, denote the nodes in V with odd degree by V_{odd} and the nodes with even degree by V_{even}, we know these satisfy the following equality:

$$\sum_{v \in V_{odd}} deg(v) + \sum_{v \in V_{even}} deg(v) = 2|E|.$$

We know $\sum_{v \in V_{even}} deg(v)$ and $2|E|$ are even, so $\sum_{v \in V_{odd}} deg(v)$ is even, and it can be expressed as a sum of odd numbers, thus the size of V_{odd} must be even. So we know we can get a perfect matching to pair all these odd degree nodes. Moreover, from Section 2.5, we know the minimum cost perfect matching can found in polynomial time. Since the costs are metric and optimal TSP tour can be decomposed into two perfect matchings, then the cost of one minimum cost perfect matching $c(M^*) \leq \frac{1}{2}c(OPT_{TSP})$. So for the cost for the Euler tour:

$$\begin{aligned} c(G_E) &= c(T) + c(M^*) \\ &\leq c(OPT_{TSP}) + \frac{1}{2}c(OPT_{TSP}) = \frac{3}{2}c(OPT_{TSP}). \end{aligned}$$

Then the cost of T_G is less than or equal to $\frac{3}{2}$.

Chapter 3

Circulations applied to 2ECSS

In this chapter we introduce the basic definitions and theorems for circulations. We provide an idea for an approximation algorithm to compute a 2-edge-connected spanning subgraph based on circulations. In the last two sections, we introduce Huh's approximation algorithm [Huh04] for k-edge-connected graphs, and present our new approximation algorithm based on circulations which improves upon Huh's [Huh04] result both in ratio and simplicity.

3.1 Background on Circulations

Given a directed graph $D = (V, A)$ and two nodes $s, t \in V$, a vector $f \in \mathbb{R}^A$ is called a *flow* from s to t if it satisfies the following constraints:

$$f(\delta^{in}(v)) = f(\delta^{out}(v)) \text{ for all } v \in V \setminus \{s, t\}. \tag{3.1}$$

Constraints (3.1) are called the *flow conservation constraints*, s is called the *source* and t is the *sink*. For an arc $e \in A$, f_e is called the *flow* of e. Given *arc capacities*

$u \in \mathbb{R}^A$, if f satisfies the non-negative constraints and capacity constraints:

$$0 \leq f_a \leq u_a \text{ for all } a \in E, \tag{3.2}$$

then we say f is a *feasible flow*.

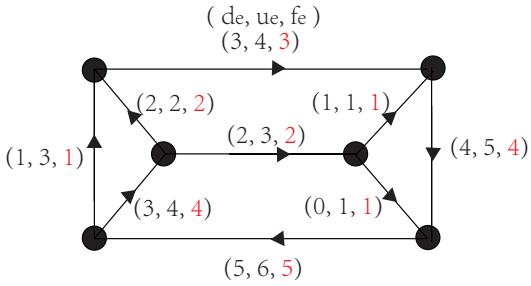


FIGURE 3.1: Graph which shows a feasible circulation under given demands and capacities.

A *circulation* f is a variation of flow which satisfies all the constraints for a flow but it does not have a source or sink. For a directed graph $D = (V, A)$ and $f \in \mathbb{R}^A$, f is a *circulation* for D if $f(\delta^{in}(v)) = f(\delta^{out}(v))$ for all $v \in V$. Given arc demands $d \in \mathbb{R}^A$, arc capacities $u \in \mathbb{R}^A$ and a circulation $f \in \mathbb{R}^A$, if for every $e \in E$, f_e satisfies $d_e \leq f_e \leq u_e$, then we say f is a *feasible* circulation (see Figure 3.1). Finally, given arc costs $c \in \mathbb{R}^A$, the ILP of the *minimum cost circulation problem* is as follows:

$$\begin{aligned} &\text{minimize } cf \\ &\text{subject to } f(\delta^{in}(v)) = f(\delta^{out}(v)) \text{ for all } v \in V, \\ &\qquad d_e \leq f_e \leq u_e \text{ for all } e \in A, \\ &\qquad f_e \text{ integer.} \end{aligned} \tag{3.3}$$

If we take out the integer constraints, we would get the LP relaxation for the

minimum cost circulation problem:

$$\begin{aligned} &\text{minimize } cf \\ &\text{s.t } \ f(\delta^{in}(v)) = f(\delta^{out}(v)) \text{ for all } v \in V, \\ &\qquad d_e \leq f_e \leq u_e \text{ for all } e \in A. \end{aligned} \tag{3.4}$$

Given a digraph $D = (V, A)$, edge demands $d \in \mathbb{R}^A$ and capacities $u \in \mathbb{R}^A$, the circulation polytope is defined by the constraints of (3.4), i.e.

$$\begin{aligned} &f(\delta^{in}(v)) = f(\delta^{out}(v)) \text{ for all } v \in V, \\ &d_e \leq f_e \leq u_e \text{ for all } e \in A. \end{aligned} \tag{3.5}$$

Next we present some well-known relevant theorems.

Theorem 3.1.1. *(Hoffman's Circulation Theorem [Hof60]).*
Given a digraph $D = (V, A)$, arc demands and capacities $d, u \in \mathbb{R}^A$ with $d_e \leq u_e$ for all $e \in A$, there exists a circulation f satisfying $d_e \leq f_e \leq u_e$ if and only if

$$d(\delta^{in}(U)) \leq u(\delta^{out}(U)) \tag{3.6}$$

for each subset $U \subset V$. Moreover, if d and u are integer, then there exists a feasible circulation that is integer.

This implies that for any given circulation problem, if we want to prove there exists a feasible circulation, then we can either show that (3.6) is satisfied, or manually set a flow assignment and prove that it is a feasible circulation.

Theorem 3.1.2. *Given a circulation polytope P as in (3.5), if d and u are integer-valued, then P is an integer polytope [Sch95].*

Corollary 3.1.3. *Given digraph $D = (V, A)$ with costs $c \in \mathbb{R}^A$, and integer demands $d \in \mathbb{Z}^A$ and capacities $u \in \mathbb{Z}^A$, if there exists a feasible circulation $f' \in \mathbb{R}^A$ (which*

might be fractional) for the minimum cost circulation problem defined in (3.4), then there exists an integer circulation f with cost $cf \leq cf'$.

Proof. This follows directly from Theorem 3.1.2 and Theorem 2.3.1. □

In 1984, Tardos [Tar85] gave a strongly polynomial time algorithm for minimum cost circulation problem, showing it can be found in $O(n^2m^3 \log n)$ time. Moreover, it is showed that if d and u are integral, then an integral circulation can be found.

Corollary 3.1.4. *Let $D = (V, A)$ be a digraph with costs $c \in \mathbb{R}^A$, and integer demands $d \in \mathbb{Z}^A$ and capacities $u \in \mathbb{Z}^A$. If there exists a feasible circulation $f' \in \mathbb{R}^A$ (which might be fractional) for the minimum cost circulation problem, then we can find an integer circulation f with cost cf less than or equal to cf' in polynomial time.*

Proof. From Corollary 3.1.3, we know there must exist an integer circulation f'' with cost $cf'' \leq cf'$. Moreover, since d and u are integral, Tardos's Algorithm [Tar85] shows that we can find an integral minimum cost circulation f in polynomial time, and we know the the cost for minimum cost circulation is no more than the cost for any feasible circulation. Thus $cf \leq cf'' \leq cf'$. From the above we know we can polynomially find an integer circulation f with cost $cf \leq cf'$.

So Corollary 3.1.4 is proved. □

3.2 Using Circulations for 2ECSS

In this section, we will show how we can apply circulations to the 2ECSS problem for a given 2-edge-connected graph $G = (V, E)$. The main idea is to start with a depth first search tree T of G, then add some back edges to form a 2-edge-connected spanning subgraph. Because we know every node is incident with at least one tree edge, it follows that every non-empty cut must cross at least one tree edge. Therefore all the cuts can be classifies into two situations:

1. Cuts which contain exactly one tree edge.

2. Cuts which contain at least two tree edges.

If we want to get a 2-edge-connected spanning subgraph for G, we know we only need to take care of situation (1), which means we need to ensure that every tree cut in the depth first search tree contains at least one back edge, thus giving a total of at least two edges in any tree cut. From the property of depth first search tree, there is no edge between branches, so the back edges to pair the cut contains one tree edge would only rely on the back edges in the corresponding tree cut. If we use circulations, we can set the demand for each tree edge to be 1, and the capacity of each back edge to be 1, then the support graph of an integer-valued circulation would form a 2-edge-connected spanning subgraph of G. Moreover, if we want to ensure that the number of edges we add is minimized, we can set the cost for all the back edges to be 1, and for tree edges to be 0.

Here we present the idea for the algorithm to approximate 2ECSS using circulations in more details.

Algorithm 3.1. CirFor2ECSS

Input: A 2-edge-connected graph $G = (V, E)$.

Output: A 2-edge-connected spanning subgraph H of G.

Step 1 : Grow a depth first search tree T in G, denote the graph with orientation as $D = (V, A)$.

Step 2 : Define a minimum cost circulation problem on D as follows:

$$d_e = \begin{cases} 1 \text{ for } e \in T, \\ 0 \text{ otherwise,} \end{cases} \qquad u_e = \begin{cases} 1 \text{ for } e \in B, \\ \infty \text{ otherwise,} \end{cases} \qquad c_e = \begin{cases} 1 \text{ for } e \in B, \\ 0 \text{ otherwise.} \end{cases} \tag{3.7}$$

Step 3 : Solve the problem to get an integer minimum cost circulation f.

Step 4 : Return the support graph H of f as the 2-edge-connected spanning subgraph.

We will prove the two main theorems below to show that Algorithm 3.1 works.

Theorem 3.2.1. *Given the circulation problem defined in (3.7), and a feasible solution f, we will show there exists an integer circulation for which the support graph has at most $n-1+cf$ edges .*

Proof. First we get the depth first search tree rooted at node r in graph G. Using the notation in Section 2.4, we denote the tree edges by T and the back edges by B. The tree edges are directed away from the root r, and the back edges are directed towards the root r. Then we form the directed graph $D = (V, A)$ by setting $A = T \cup B$. Note that instead of using infinity for u_e with $e \notin B$ in the circulation problem (3.7), we can set u_e equal to a very large integer number. We can see that all u and d are integer, thus from Theorem 3.1.3, we know there must exist an integer circulation f' with cost less than equal to cf. So we form a new graph H by taking the support graph of f'. This support graph consists of all edges in T and all back edges $e \in B$ with $f'_e = 1$. Thus the number of edges in H equals $n-1+cf' \leq n-1+cf$.

Hence Theorem 3.2.1 is proved. □

Theorem 3.2.2. *Given a graph $G = (V, E)$, and a feasible circulation f for the circulation problem (3.7), we can get a 2-edge-connected spanning subgraph for G with the number of edges at most $n-1+cf$.*

Proof. From Theorem 3.2.1, we know there must exist another integer circulation f' for which the support graph H has at most $n-1+cf$ edges. Since the integer circulation f' must satisfy the demand constraints, every tree edge e must have $f'_e \geq 1$. Note that every back edge b_i forms a unique cycle C_i in T, so the flow on every tree edge t_i depends on $\sum f_{b_i}$ where C_i denotes the cycles containing t_i. Thus for every tree cut, we have at least a flow of one coming in and leaving out. This implies that

$f_e = 1$ for at least one back edge e in the tree cut. Then for the undirected graph, we have at least two edges, i.e. $\delta(S) \geq 2$ for all $S \subset V$. So the support graph for f' is a 2-edge-connected spanning subgraph.

So Theorem 3.2.2 is proved. □

3.3 Huh's Results on 2ECSS

Given any 2-edge-connected graph $G = (V, E)$, we know G contains an ear decomposition from Theorem 2.6.1. If we delete all the trivial ears and obtain a valid ear decomposition of the spanning subgraph H of G, then we know H is a 2-edge-connected spanning subgraph of G. To minimize the number of edges in H, we need to maximize the number of trivial ears or minimize the number of non-trivial ears in the ear decomposition. With the above idea, Huh [Huh04] presented an approximation algorithm for 2ECSS. He gave a recursive algorithm to find such an ear decomposition, each iteration to construct a non-trivial ear. Then he upper-bounded the number of ears using a potential function, and achieved the following theorems:

Theorem 3.3.1. *Given a k-edge-connected graph $G = (V, E)$, where $k \geq 2$, $|V| = n$ and $|E| = m$, the algorithm will finds a 2-edge-connected spanning subgraph with the number of edges at most*

$$n + \frac{m-n}{k-1}.$$

Corollary 3.3.2. *Let $G = (V, E)$ be a k-regular k-edge-connected graph, where $k \geq 2$, $|V| = n$ and $|E| = m$, the algorithm will finds a 2-edge-connected spanning subgraph with the number of edges at most*

$$(1 + \frac{k-2}{2k-2})n.$$

Moreover, if k=3, 4, 5 and 6, this algorithm ensures approximation ratio of $\frac{5}{4}$, $\frac{4}{3}$, $\frac{11}{8}$

and $\frac{7}{5}$, respectively.

3.4 Improvement on Huh's Result for 2ECSS

In this section we demonstrate the usefulness of circulations for 2ECSS by presenting an approximation algorithm that improves on Huh's result [Huh04] for the 2ECSS problem on k-edge-connected graphs, both in value and the simplicity of the proof.

Based on our own algorithm for 2ECSS using circulations from Section 3.2, we present the following theorem:

Theorem 3.4.1. *Given any k-edge-connected graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, there exist a 2-edge-connected spanning subgraph with the number of edges at most*

$$n + \frac{m-n}{k-1} - \frac{k-2}{k-1}$$

For $k \geq 2$.

Proof. We apply Algorithm 3.1 to prove this theorem. First we will grow a depth first search tree in G, and get the corresponding directed graph $D = (V, A)$. Denote the set of back edges across the tree cut through edge $e \in T$ by δ_{T_e}, then we would get a feasible circulation y by setting the flow as follows:

$$y_e = \begin{cases} \frac{1}{k-1} \text{ for all } e \in B, \\ \sum \frac{1}{k-1} |\delta_{T_e}| \text{ for all } e \in T. \end{cases} \tag{3.8}$$

First we need to show the circulation in (3.8) is a feasible circulation. To show the circulation is feasible, we need to show it satisfies the three constraints in (3.4). For the flow conservation constraints: Since we set this circulation by assigning flow $\frac{1}{k-1}$ to every back edge, and get the corresponding flow in the tree edge, thus it satisfies the flow conservation constraint.

For the demand constraints and capacity constraints: It is clear this flow satisfies the non-negativity constraint on edges. Also, because the capacity for back edges is 1 and for tree edges is infinite, this assignment satisfies the capacity constraint. Also, this is a k-edge-connected graph, therefore for every edge cut we have at least k-edges. Then for every tree cut, $|\delta_{T_e}| \geq (k-1)$ must hold for all $e \in T$, so the flow on every tree edge must be at least 1, thus it satisfies the demand constraints for edges in T. From the above analysis, we know y is a feasible circulation. From Theorem 3.2.2, we know there must exist a 2-edge-connected spanning subgraph with cost t edges, where

$$\begin{aligned} t \leq n-1+c(y) &= n-1+\frac{|B|}{k-1} \\ &= n-1+\frac{(|E|-(n-1))}{k-1} \\ &= n+\frac{m-n}{k-1}-\frac{k-2}{k-1}. \end{aligned}$$

So Theorem 3.4.1 is proved. □

Corollary 3.4.2. *Given a k-regular k-edge-connected graph $G = (V, E)$, $|V| = n$, $|E| = m$, we can find a 2-edge-connected spanning subgraph with at most*

$$(1+\frac{k-2}{2k-2})n-\frac{k-2}{k-1}$$

edges.

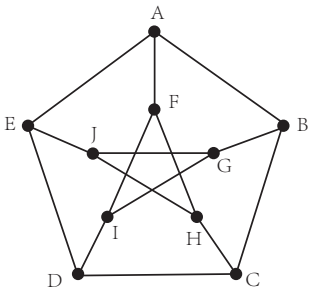
Proof. We know for a k-regular k-edge-connected graph, the number of edges m in the graph is $\frac{kn}{2}$, thus using the result from Theorem 3.4.1, the number of edges in the 2-edge-connected spanning subgraph is at most

$$\begin{aligned} n+\frac{(m-n)}{k-1}-\frac{(k-2)}{k-1} &= n+\frac{(\frac{kn}{2}-n)}{k-1}-\frac{(k-2)}{k-1} \\ &= \frac{3k-4}{2k-2}n-\frac{(k-2)}{k-1}. \end{aligned}$$

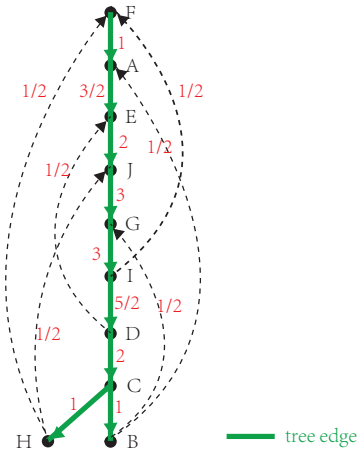
Hence Corollary 3.4.2 is proved. □

Note that we also have a $\frac{3}{2}$-approximation algorithm of 2ECSS for k-edge-connected graphs, since n is a lower bound for 2ECSS. If we apply Gabow and Tarjan's [GT91] algorithm to solve the minimum cost circulation problem, then Algorithm 3.1 runs in $\Theta(n^{\frac{3}{4}} m \log N)$ where N denotes the largest magnitude of a cost.

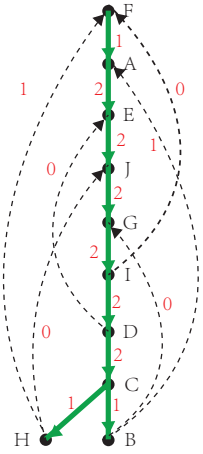
Example In Figure 3.2, we apply our algorithm to the Peterson graph to get a 2-edge-connected spanning subgraph. In Figure 3.2(b) we get a feasible circulation in the depth first search tree we obtained, then we get an integer circulation with smaller cost in 3.2(c). We know the Peterson graph is a 3-regular 3-edge-connected graph, so from Corollary 3.4.2, we can get a 2-edge-connected spanning subgraph with at most 12 edges, and 3.2(d) verifies the result.



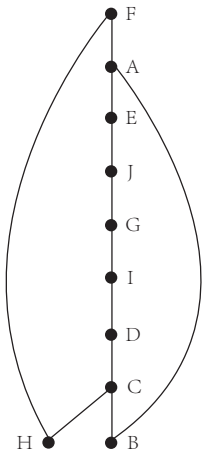
(a) Peterson Graph is a cubic 3-edge-connected graph, denote it as G.



(b) Grow a depth first search tree. Then assign flow on every edge as in (3.8) to get a feasible circulation f with cost cf equals 3.



(c) An integer circulation f^* with cost at most cf, the cost of f^* is 2.



(d) The final 2-edge-connected spanning subgraph H with 11 edges. E(H)=11n/10.

FIGURE 3.2: Graph illustrating our method for 2ECSS.

Chapter 4

T-joins and Special Vectors

In this chapter, we focus on the polytopes for T-joins. In Section 4.1 we investigate two polytopes of T-joins, and show the equivalence of these two polytopes under certain conditions. In Section 4.2, we study the one-third vector that was used in [SV12] for their approximation algorithm for graph TSP. Then given a new result, which is to show a one-third two-third vector is in the T-join polytope for certain graphs, which we think might be useful for graph TSP approximation in the future.

4.1 T-join

Given a graph $G = (V, E)$ and a node set $T \subseteq V$, then an edge set $F \subseteq E$ is a *T-join* if T exactly equals the odd degree nodes in the subgraph $G' = (V, F)$. It is easily seen that if there exists a T-join in G, the size of T must be even. A *T-cut* is an edge cut which separates T into two sets of odd size. Let J denote the set of all the odd degree nodes in G, then we call such a J-join an *odd join* for G (see Figure 4.1 for example). Note that for any graph $G = (V, E)$, E itself is an odd join for G. Given non-negative edge costs $c \in \mathbb{R}^E$, $c_e \geq 0$ for all $e \in E$, the LP formulation for the

minimum cost T-join problem can be written as follows:

$$\begin{aligned} &\text{Minimize } cx \\ &\text{subject to } x(D) \geq 1, \text{ for all } T\text{-cuts } D, \\ &0 \leq x_e \leq 1 \text{ for all } e \in E. \end{aligned} \tag{4.1}$$

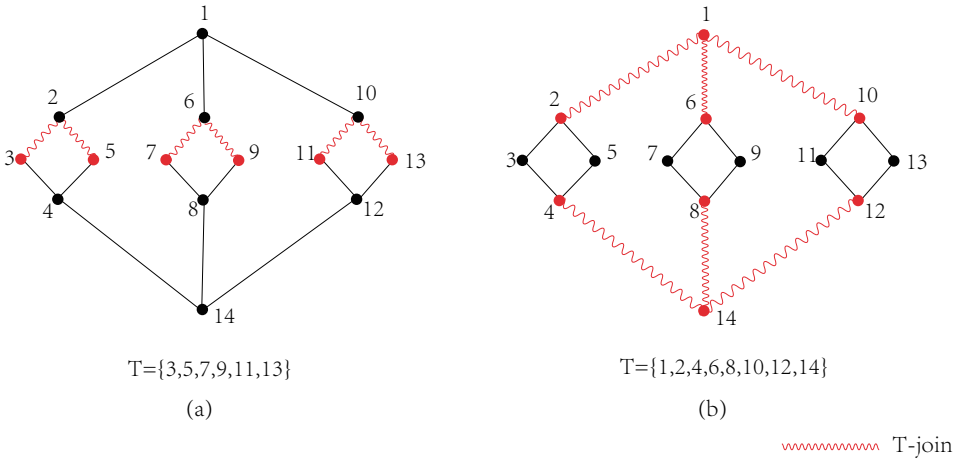


FIGURE 4.1: (a) Shows a possible T-join when we set T equal to all the even degree nodes in the graph. (b) Shows a possible T-join when we set T equal to all the odd degree nodes in the graph, thus the T-join in (b) is an odd join for G.

If the edge costs c contain negative costs, then the LP formulation for the minimum cost T-join problem is defined as follows [Sch03]:

$$\begin{aligned} &\text{Minimize } cx \\ &\text{subject to } x(\delta(U) \setminus F) - x(F) + |F| \geq 1 \text{ for all } U \subset V, \\ &\qquad F \subseteq \delta(U) \text{ with } (|U \cap T| + |F|) \text{ odd }, \\ &0 \leq x_e \leq 1 \text{ for all } e \in E. \end{aligned} \tag{4.2}$$

The T-join polytope is determined by the following inequalities:

$$\begin{aligned} x(\delta(U) \setminus F) - x(F) + |F| \geq 1 \text{ for all } U \subset V, \\ F \subseteq \delta(U) \text{ with } (|U \cap T| + |F|) \text{ odd }, \\ 0 \leq x_e \leq 1 \text{ for all } e \in E. \end{aligned} \tag{4.3}$$

Next we will prove the T-join polytope (4.3) is equivalent to the following polytope described in [SV12], under certain conditions.

Theorem 4.1.1. *Given a graph $G = (V, E)$, let T denotes all the odd degree nodes in G. Then the following inequalities define the same polytope as the T-join polytope (4.3):*

$$\begin{aligned} x(\delta(U) \setminus F) + |F| - x(F) \geq 1 \quad \textit{for all } U \subset V \textit{ and } F \subseteq \delta(U) \\ \textit{with } |\delta(U) \setminus F| \textit{ odd}, \\ 0 \leq x_e \leq 1 \textit{ for all } e \in E. \end{aligned} \tag{4.4}$$

Proof. The polytope defined here is slightly different from the one defined above, as it requires $|U \cap T| + |F|$ to be odd, while here it requires $|\delta(U) \setminus F|$ to be odd. So we just need to show these two parts have the same parity when T equals all the odd degree nodes in G.

Recall $\lambda(U)$ denotes the edges that have both ends in U. Then

$$\begin{aligned} |\delta(U)| &= \sum_{v \in U} d_v - 2 * |\lambda(U)| \\ &= \sum_{v \in U \cap T} (d_v) + \sum_{v \in U \setminus T} d_v - 2 * |\lambda(U)|. \end{aligned}$$

Since all nodes in $U \setminus T$ have even degrees, $\sum_{v \in U \setminus T} d_v - 2 * |\lambda(U)|$ must be an even number. Therefore $|\delta(U)|$ has the same parity as $|U \cap T|$. So if $|U \cap T| + |F|$ is odd, $|\delta(U)| + |F|$ must also be odd. This means $|\delta(U)|$ and $|F|$ have different parities. We

can see $|\delta(U) \setminus F|$ is also odd.

From the above analysis we know that if we set T to be all the odd degree nodes in the graph, these two polytopes are the same.

Hence Theorem 4.1.1 is proved. □

In 1986, Karzanov [Kar86] gave an $\Theta(|T|m \log n + |T|^3 \log |T|)$ time algorithm to find a minimum cost T-join.

4.2 Special Vectors in the T-join Polytope

For the T-join polytope, there are some special vectors that satisfy all the constraints for certain graphs. It is known that the T-join polytope is an integer polytope [Sch03], thus we know any feasible point x in the T-join polytope can be expressed as a convex combination of incidence vectors of T-joins. Applying Theorem 2.3.1, we know there exists a T-join with cost at most cx. In this section we introduce some of these special vectors in the T-join polytope.

4.2.1 One Third Vector

For this section, given a graph G, we define the *one third vector for* G to be the vector $x^* \in \mathbb{R}^E$ defined by $x^*_e = \frac{1}{3}$ for all $e \in E$.

Theorem 4.2.1. *Given a 2-edge-connected graph $G = (V, E)$, then the one third vector x^* for G defined by $x^*_e = \frac{1}{3}$ for all $e \in E$ is in the T-join polytope (4.4), where T denotes all the odd degree nodes in G*[1].

Proof. We will prove the vector x^* satisfies the constraints in (4.4).

[1]This theorem proves a result stated and used in [SV12] page 21 without proof.

- For the first constraint, we have:

$$\begin{aligned} x^*(\delta(U) \setminus F) - x^*(F) + |F| &= \frac{1}{3}(|\delta(U) \setminus F|) - \frac{1}{3}|F| + |F| \\ &= \frac{1}{3}(|\delta(U) \setminus F|) + \frac{2}{3}|F| \\ &= \frac{1}{3}(|\delta(U)| + |F|). \end{aligned} \tag{4.5}$$

  Since $|\delta(U) \setminus F|$ is odd, we have the following two scenarios:

  1. $|F|$ is odd, i.e. $|F| \geq 1$ and $|\delta(U)|$ must be even and must be a non-zero number, thus $|\delta(U)| \geq 2$ must hold. Then we have $|F| + |\delta(U)| \geq 3$.
  2. $|F|$ is even, i.e. $|F| \geq 0$ and $|\delta(U)|$ must be odd, thus $|\delta(U)| \geq 1$ holds. But since G is 2-edge-connected, $|\delta(U)|$ must greater than or equal to 3, then we also have $|F| + |\delta(U)| \geq 3$.

  From the above analysis, we know $|F| + |\delta(U)| \geq 3$ must hold, then combined with the equation in (4.5), we have $\frac{1}{3}(|F| + |\delta(U)|) \geq 1$ holds, so $x^*(\delta(U) \setminus F) - x^*(F) + |F| \geq 1$ for all $U \subset V(G)$ and $F \subseteq \delta(U)$ with $|\delta(U) \setminus F|$ odd .

- For the second constraint, since $x^*_e = \frac{1}{3}$ for all $e \in E$, thus $0 \leq x^*_e \leq 1$ holds.

The vector x^* satisfies all the constraints for the T-join polytope, so it is in this polytope.

Hence Theorem 4.2.1 is proved. □

Corollary 4.2.2. *Given a 2-edge-connected graph $G = (V, E)$, then the vector x^* is in the T-join polytope in (4.3) where T denotes all the odd degree nodes in G.*

Proof. From Theorem 4.1.1 we know that if T equals all the odd degree nodes in G, those two polytopes are the same. From Theorem 4.2.1 we know the vector x^* is in (4.4), thus it is also in (4.3). So Corollary 4.2.2 is proved. □

4.2.2 Analysis of $\frac{1}{3}, \frac{2}{3}$-vector and Special Half-integer Vector

Given a graph $G = (V, E)$, and a partition $A,\ B$ of E that $A \subseteq E,\ B \subseteq E,\ A \cap B = \emptyset,\ A \cup B = E$, we define a vector x' as follows,:

$$x'_e = \begin{cases} \frac{1}{3},\ e \in A, \\ \frac{2}{3},\ e \in B. \end{cases} \tag{4.6}$$

We have the following theorem for x':

Theorem 4.2.3. *If G is 3-edge-connected, then the vector x' as defined in (4.6) above is in the T-join polytope defined in (4.3) where T denotes all the odd degree nodes in G.*

Proof. Let us partition $\delta(U)$ into $\delta_1(U)$ and $\delta_2(U)$ and partition F into F_1 and F_2, where $\delta_1(U)$, $\delta_2(U)$ denotes the set of $\frac{1}{3}$-edges and $\frac{2}{3}$-edges in $\delta(U)$, respectively, and F_1, F_2 denote the set of $\frac{1}{3}$-edges and $\frac{2}{3}$-edges in F, respectively. For the first constraint in the T-join polytope, let us do the following transformation to the left side of the inequality:

$$\begin{aligned} x'(\delta(U) \setminus F) + |F| - x'(F) &= \frac{1}{3}|\delta_1(U) \setminus F_1| + \frac{2}{3}|\delta_2(U) \setminus F_2| + |F| - (\frac{1}{3}|F_1| + \frac{2}{3}|F_2|) \\ &= \frac{1}{3}|\delta_1(U) \setminus F_1| + \frac{2}{3}|\delta_2(U) \setminus F_2| + \frac{2}{3}|F_1| + \frac{1}{3}|F_2| \\ &= \frac{1}{3}|\delta(U)| + \frac{1}{3}|F_1| + \frac{1}{3}|\delta_2(U) \setminus F_2|. \end{aligned} \tag{4.7}$$

Since G is 3-edge-connected, $|\delta(U)| \geq 3$ always hold, which also implies $\frac{1}{3}|\delta(U)|$ is greater than or equal to 1, so $\frac{1}{3}|\delta(U)| + \frac{1}{3}|F_1| + \frac{1}{3}|\delta_2(U) \setminus F_2| \geq 1$ is valid.

It is easy to see that for every $e \in E$, the x'_e value satisfies $0 \leq x'_e \leq 1$. From the above analysis, we know the vector x' is in the T-join polytope.

Hence Theorem 4.2.3 is proved. □

If G is 2-edge-connected, the vector x' may not be in the T-join polytope. From Theorem 4.1.1 we know if T denotes all the odd degree nodes in G, the T-join polytope in (4.3) equals the polytope in (4.4). Then we can use the condition in (4.4), i.e. $|\delta(U) \setminus F|$ odd. Consider the following example: Given a 2-edge-connected graph G, let $|\delta(U)| = 2,\ |U_1| = 1,\ |U_2| = 1,\ |F_1| = 0,\ |F_2| = 1$. So we know $|\delta(U) \setminus F|$ equals to one, thus is odd, satisfying the condition, but $x(\delta(U) \setminus F) + |F| - x(F) = \frac{2}{3}$, which is less than one. So under this scenario, x' is not in the T-join polytope. However, under certain other conditions, even if G is 2-edge-connected, the vector x would still be in T-join polytope.

Theorem 4.2.4. *If G is cubic and 2-edge-connected, and set A defined in (4.6) above contains all the edges which are in a 2-edge cut, then the vector x' is in the T-join polytope defined in (4.3) for $T = V$.*

Proof. From Theorem 4.2.3, we know that if we want to prove x is in the T-join polytope, then the result of (4.7) being greater than or equal to one must hold, i.e.

$$\frac{1}{3}|\delta(U)| + \frac{1}{3}|F_1| + \frac{1}{3}|\delta_2(U) \setminus F_2| \geq 1. \tag{4.8}$$

Since G is 2-edge-connected, so $|\delta(U)| \geq 2$ for all $U \subset V$. Then we have the following two situations:

1. $|\delta(U)| \geq 3$. It is easy to see that equation (4.8) holds.

2. $2 \leq |\delta(U)| < 3$, i.e. $|\delta(U)| = 2$. According to the construction of the vector x', both edges in this cut hold value $\frac{1}{3}$.

   Since G is cubic, every node in V has odd degree, so T equals V. According to Theorem 4.1.1, we can use the polytope described in (4.4). Then $|\delta(U) \setminus F|$ must be odd, if we require inequality (4.8) to hold. Therefore in this case, $|\delta(U)|$ is even, and $|F|$ must be odd, i.e. $|F| = 1$. Because we know all the

edges in this cut have the value $\frac{1}{3}$, so if there is an edge in F, it must in F_1. So $\frac{1}{3}|\delta(U)| + \frac{1}{3}|F_1| + \frac{1}{3}|\delta_2 \setminus F_2| = \frac{1}{3} * 2 + \frac{1}{3} * 1 = 1$. Equation (4.8) still holds.

According to the above analysis, we know the vector x' is in the T-join polytope.

So Theorem 4.2.4 is proved. □

Corollary 4.2.5. *Let $G_{x^*} = (V, E)$ be the support graph of a special half-integer solution x^*(see Section 2.3). Define a vector $x' \in \mathbb{R}^E$ as follows:*

$$x'_e = \begin{cases} \frac{1}{3}, \ e \text{ is a 1-edge for } x^*, \\ \frac{2}{3}, \ e \text{ is a } \frac{1}{2}\text{-edge for } x^*. \end{cases} \tag{4.9}$$

Then x' is also in the T-join polytope for $T = V$.

Proof. We will prove the vector defined in (4.9) contained in the vector defined in Theorem 4.2.4, i.e. every edge e in a 2-edge cut has $x'_e = \frac{1}{3}$. Since we know G_{x^*} comes from a half-integer solution, then we know for any 2-edge cut $\delta(U)$ with $U \subseteq V$, both edges in $\delta(U)$ must be 1-edge. Thus we know all edges in a 2-edge cut has $x'_e = \frac{1}{3}$. From Theorem 4.8, we know x' is in the T-join polytope.

So Corollary 4.2.5 is proved. □

Chapter 5

Applying T-joins and Circulations to Graph TSP

In this chapter, we introduce the definition of removable pairings in Section 5.1, which is very useful in what follows. We also provide a proof of results in [MS11] that we make simpler and clearer by combining results found in [MS11], [SV12] that shows how we get a TSP tour with certain cost using removable pairs. In Section 5.2, we show how Mömke and Svensson [MS11] get removable pairings from a depth first search tree, and introduce some relevant theorems stated in [MS11]. In Section 5.3, we describe the method to get removable pairings from ear decomposition stated in [SV12].

5.1 Removable Pairings

As described in Section 2.7, if we want to get an Eulerian graph, one way is to find a spanning tree first, then add the matchings or joins to correct the parity of the odd degree nodes in the tree. But we can also find a way to remove some edges to correct those wrong parities by using the method of removable pairings. This method was used by Mömke and Svensson in [MS11] for graph TSP.

As defined in [MS11], given a 2-vertex-connected graph $G = (V, E)$, a tuple $\{R, P\}$ which consists of a removable set of edges R and a set of pairs of edges P is called a *removable pairing* if it satisfies the following constraints:

- $R \subseteq E$.
- P is a set of edge pairs (e_1, e_2) with $e_1, e_2 \in R$, and such that e_1 and e_2 have a common endpoint with degree at least three.
- For any edge $e \in R$, e belongs to at most one pair in P.
- If we delete a subset R' of edges in R such that R' contains at most one edge for any pair in P, the graph stays connected.

To begin we present the following lemma.

Lemma 5.1.1. *Let $G = (V, E)$ be any graph, and J be an odd join for G. Let K be any subset of E. Then the multigraph $G^* = (V^*, E^*)$ where $V^* = V$ obtained by setting:*

$$E^* = E \cup (J \setminus K) - (J \cap K)$$

has even degree (possibly 0) at every node.

Proof. For the odd join J, we know it is incident with every node $v \in V$ with $|\delta(v) \cap J|$ edges, and $|\delta(v) \cap J|$ has the same parity as $|\delta(v)|$. So for any node $v^* \in V^*$,

$$\begin{aligned} |\delta(v^*)| &= |\delta(v)| - |(\delta(v) \cap J) \cap K| + |(\delta(v) \cap J) \setminus K| \\ &= |\delta(v)| - 2|(\delta(v) \cap J) \cap K| + |(\delta(v) \cap J)|. \end{aligned}$$

Since $\delta(v)$ and $|\delta(v) \cap J|$ have the same parity, thus $|\delta(v)| + |(\delta(v) \cap J)|$ is even. So for every node $v^* \in V^*$, $|\delta(v)|$ is even.

Hence Lemma 5.1.1 is proved. □

Given a 2-vertex-connected graph $G = (V, E)$ with non-negative edge costs $c \in \mathbb{R}^E$, Mömke and Svensson [MS11] prove the following nice result for the case where $c_e = 1$ for all $e \in E$, i.e. graph TSP. Here we present the simpler proof presented in [SV12], and extend the result to general edge costs using ideas from [BSvdSS11].

Theorem 5.1.2. *Given a 2-vertex-connected graph $G = (V, E)$ with non-negative edge costs $c \in \mathbb{R}^E$ and a removable pairing (R, P), we can always find an Eulerian graph with the number of edges less than or equal to $\frac{4}{3}c(E) - \frac{2}{3}c(R)$ in polynomial time.*

Proof. First we will show there exists an odd join J for G (i.e. T is the set of odd degree nodes in G) with cost less than or equal to $\frac{1}{3}c(E) - \frac{2}{3}c(R)$, and J contains at most one edge of each pair in P.

We assign a new cost on each edge as follows:

$$c'_e = \begin{cases} -c(e) \text{ for all } e \in R, \\ c(e) \text{ otherwise.} \end{cases}$$

Then we apply the gadget described in [SV12] to construct an auxiliary graph $G' = (V', E')$ as follows: For every removable pair (vw, vw') in P, add a new node v_p, then add an edge vv_p with cost $c'_{vv_p} = 0$ and replace edges vw and vw' with two edges v_pw, v_pw' with weights $c'_{vw}, c'_{vw'}$ (see Figure 5.1).

Since G is 2-vertex-connected, thus also 2-edge-connected, it follows that the auxil-

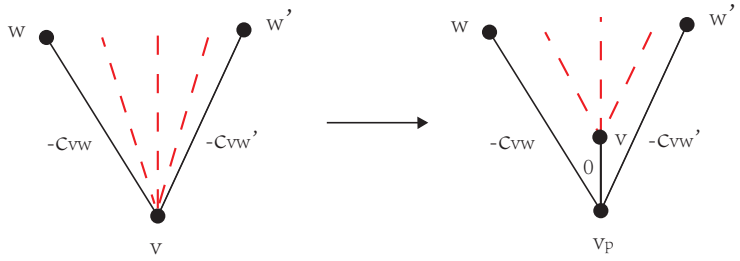


FIGURE 5.1: Graph illustrating the gadget.

iary graph G' is also 2-edge-connected (i.e. the new edge vv_p cannot be a cut edge in G', as this would imply v is a cut node in G). From Theorem 4.2.1 we know that the one-third vector x^* with $x^*_e = \frac{1}{3}$ for all $e \in E'$ is in the odd join polytope. Also, for all new nodes v_p, we have $x^*(\delta(v_p)) = 1$. This means that x^* is a convex combination of incidence vectors of odd joins for G', such that each of these joins has exactly one edge in $\delta(v_p)$ for all new nodes v_p. Then using Theorem 2.3.1, there exists an odd join J' in G' that meets each v_p node with exactly one edge, and $c'(J')$ is less than or equal to $\frac{1}{3}c'(G')$. Since we set the cost for edges in R to be negative, then the cost for graph G' is: $c'(E') = c(E \setminus R) - c(R) = c(E) - 2c(R)$. So $c'(J') \leq \frac{1}{3}(c(E) - 2c(R))$.

Next we will show how to transfer the odd join J' in G' back to an odd join J in G. Since we change the parity of the degree for v in G' when we apply the gadget, if $deg(v)$ is odd in G, then J' must intersect $\delta(v)$ with an even number of edges in G'. So there exist two situations for v_p:

1. J' intersects $\delta(v_p)$ with wv_p or $w'v_p$: To transfer back, we can just replace wv_p or $w'v_p$ with wv or $w'v$.

2. J' intersects $\delta(v_p)$ with vv_p: To transfer back, we just need to delete vv_p in J'.

In both cases we change the parity of $J' \cap \delta(v)$ in G by one, so $J \cap \delta(v)$ is odd in G (see Figure 5.2). If v is even in G, we can do the same and get the corresponding T-join J.

From the above analysis, we know there exists an odd join J in G with cost less than or equal to $\frac{1}{3}c(E) - \frac{2}{3}c(R)$ and which intersects each pair of edges in P with at most one edge.

Next we will show we can get an Eulerian graph from the odd join J. Let $G^* = (V^*, E^*)$ be the multigraph we get by setting $E^* = E + J \setminus R - J \cap R$. By Lemma 5.1.1, all nodes in G^* have even degree. Also, J intersects each pair of edges in P in at most one edge. So from the construction of G^*, we only delete edges in R and at

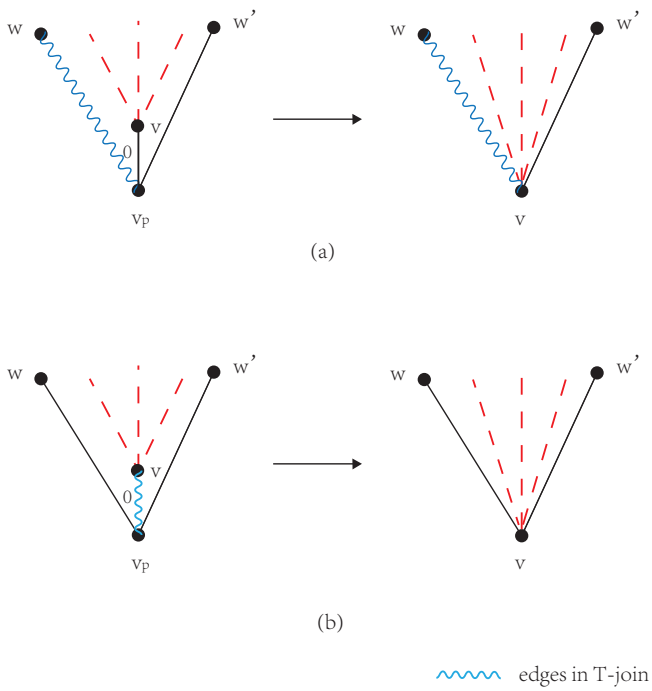


FIGURE 5.2: Graph illustrating how we transfer a T-join J' in G' to a T-join J in G, where v is odd degree in G.

most one edge in each removable pair in P, and thus it follows from the definition of removable pairings that G^* is connected. From the above analysis, we know G^* is an Eulerian graph. The cost of the edges equals $c(E) + c(J \setminus R) - c(J \cap R) = c(E) + c(J) \leq \frac{4}{3}c(E) - \frac{2}{3}c(R)$.

Hence Theorem 5.1.2 is proved. □

Corollary 5.1.3. *Given a 2-vertex-connected graph $G = (V, E)$ and a removable pairing (R, P), we can always find a TSP tour with edges less than or equal to $\frac{4}{3}|E| - \frac{2}{3}|R|$ in polynomial time for graph TSP.*

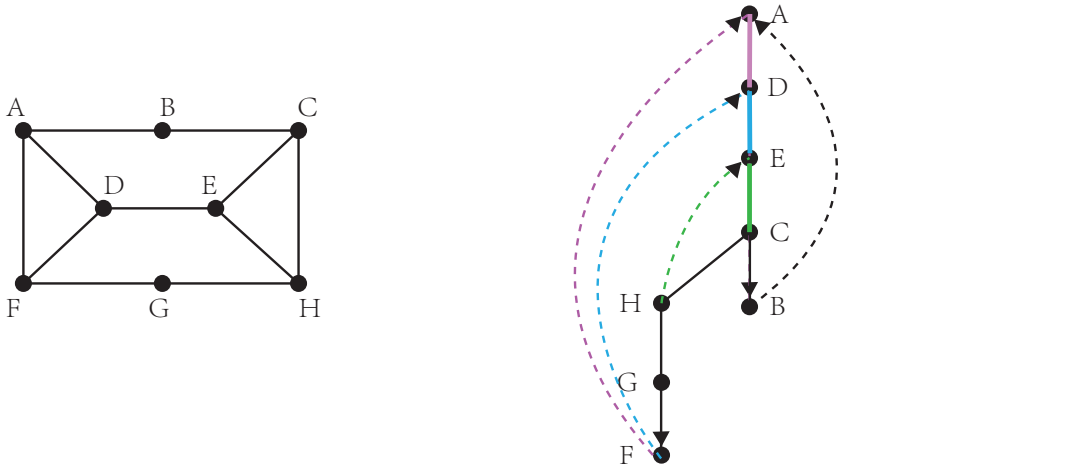
Proof. Because in graph TSP, edge cost $c_e = 1$ for all $e \in E$, thus $c(E) = |E|$, $c(R) = |R|$. From Theorem 5.1.2, we know there exists an Eulerian graph with edges less than or equal to $\frac{4}{3}|E| - \frac{2}{3}|R|$, using Theorem 2.8.1, we know there exists a TSP tour with edges less than or equal to $\frac{4}{3}|E| - \frac{2}{3}|R|$. So Corollary 5.1.3 is proved. □

5.2 Circulations used to Obtain Removable Pairings

There are different ways to find removable pairings in a graph. In the following sections we will describe the two ways currently in the literature: using depth first search tree, as was used in [MS11], and from ear-decomposition, as was used in [SV12]. To get the removable pairings from a depth first search tree from a graph G, we grow a depth first search tree T starting from any node r, then define a special circulation problem similar to the one described in Section 3.2, where we know we can get a minimum cost integer circulation f with cost $c(C)$ in polynomial time. Denote the support graph of f to be G'. Then we define the removable pairing (R, P) as follows: Let all back edges be in R, and for each tree edge $t_u = (u, v)$, if t_u has at least one back edge directed towards node u, then put t_u in R and select one edge $b_u \in B_u$ to pair with t_u, where B_u is the set of back edge coming towards u, thus (t_u, b_u) forms a pair in P. (See Figure 5.3 for example.) It can be easily seen that (R, P) satisfies the first three conditions for the definition of a removable pairing. Then we only need to prove that it also satisfies the fourth condition. We do so by proving the following theorem:

Theorem 5.2.1. *Given a graph G and a tuple (R, P) found by depth first search tree T, then for each pair $(t_u, b_u) \in P$, we can do one of the following two operations: (1) keep the edge t_u; (2) remove edge t_u add edge b_u. After we apply this to all pairs, T will become a new spanning tree T'.*

Proof. We will use induction to prove this theorem. Denote the number of pairs we swap as k. It can be easily seen that when $k = 1$, T becomes a new spanning tree. Assume when $k = i$, i.e. after we swap k pairs, T still maintains to be a spanning tree. When $k = i + 1$, let (t_u, b_u) where $t_u = (u, v),\ b_u = (w, u)$ be the $i + 1$st pair. Then after we swap i pairs, let T'_v denote the subtree rooted at v now, we know w is



(a) Original graph G.

(b) Get a removable pairing through the depth first search tree.
P={ (FA,AD), (FD,DE), (HE,EC) }.
R={ BA, (FA,AD), (FD,DE), (HE,EC) }.

FIGURE 5.3: Graph illustrating how we get a removable pairing from the depth first search tree.

in T'_v. After we swap the $i+1$st pair, u is still connected to v because the new edge we add contains node w, and w is in the subtree of v, so we know we still can get a spanning tree. So Theorem 5.2.1 is proved. □

From Theorem 5.2.1 we know if we remove at most one edge out of a pair, the graph remains connected. So we know the removable pairing we get from the depth first search tree satisfies all the conditions of a removable pairing. Another way to understand the removable pair from depth first search tree is that we can get different spanning trees by adding and removing edges.

Following the idea in Mömke and Svensson's paper [MS11], we present the next algorithm to obtain an Eulerian graph by the method of removable pairings.

Algorithm 5.1. RPForTSP

Input: A 2-vertex-connected graph $G = (V, E)$.

Output: An Eulerian graph G^*.

Step 1 : Grow a depth first search tree T in G starting at any node r and get the corresponding removable pairing (R, P).

Step 2 : Find the corresponding minimum cost odd join J.

Step 3 : Return the Eulerian graph $G^* = E - J \cap R + J \setminus R$.

Given a 2-vertex-connected graph $G = (V, E)$, Mömke and Svensson [MS11] use a circulation problem to get a removable pairing, and use it as part of their algorithm for graph TSP.

Define the tree edges as T and back edges as B, denote the graph with orientation $D = (V, A)$ with $A = T \cup B$. Then we apply the following gadget [1] to D. In order to ensure the subgraph we obtain is 2-vertex connected:
For each non root node $v \in V$ with l children $w_1, w_2 ... w_l$, replace each edge $vw_i,\ i \in \{1, ..., l\}$ with vv_i and $v_i w_i$ by adding a node v_i. Then redirect all back edges pointed toward v from the subtree rooted at w_i to v_i (See Figure 5.4). After applying the gadget, all the back edges will be directed into new nodes or the root. Denote the set of nodes with incoming edges to be τ. Let the new graph be $D' = (V', A')$, the corresponding tree edges and back edges be T' and B'.

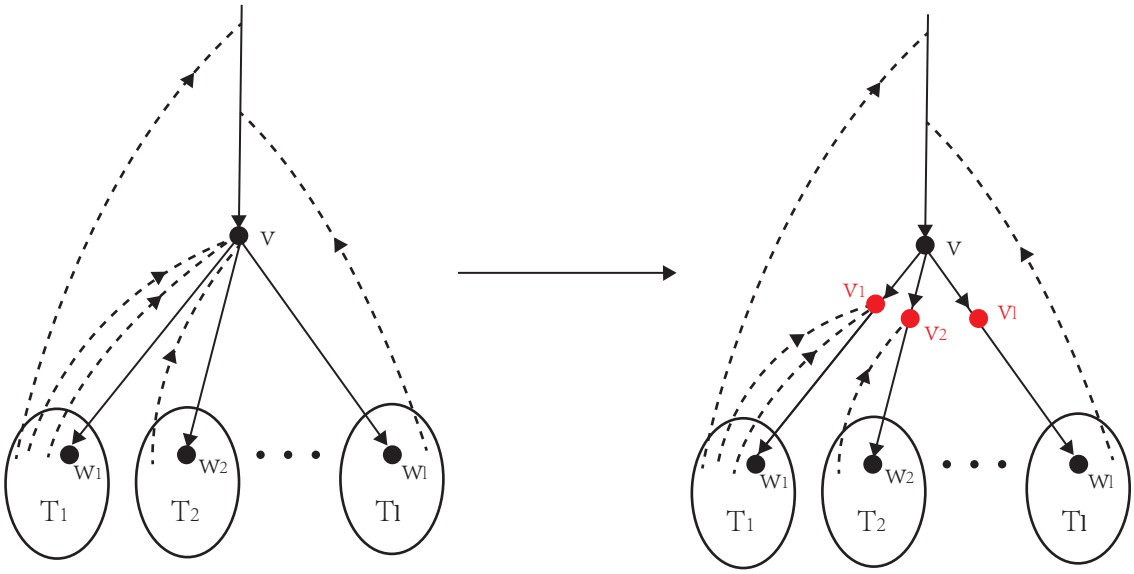


FIGURE 5.4: Mömke Svensson's Gadget.

Define a minimum cost circulation problem C in D', where the demands, capacities

[1] We call it the " Mömke Svensson Gadget ".

are defined as follows:

$$d_e = \begin{cases} 1 \text{ for } e \in T, \\ 0 \text{ otherwise,} \end{cases} \qquad u_e = \infty \text{ for all } e \in E. \tag{5.1}$$

Because only node $v \in \tau$ has incoming back edges, and only one back edge can be paired with the corresponding tree edge, to make the following analysis easier, we can define the total cost of circulation as follows:

$$c(C) = \sum_{v \in \tau} max[(f(B(v)) - 1), 0]. \tag{5.2}$$

Instead of setting an alternative cost function, we can also redefine the circulation problem by adding the gadget shown in Figure 5.5. For each node $v_i \in \tau$ with more than one incoming back edges, we add a new node v_i', then redirect all back edges to v_i', and keeping their demands and capacities. Next we add two parallel back edges $v_i'v_i$, and set one edge with cost 0, capacity 1, the other edge with cost 1, and capacity infinite.

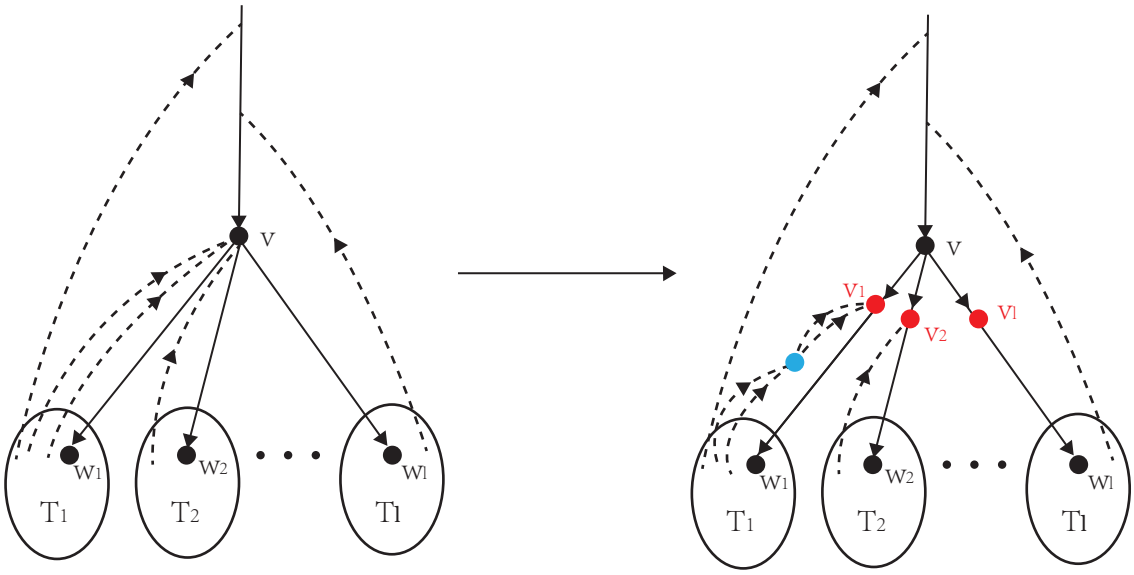


FIGURE 5.5: Graph illustrating the circulation problem equivalent to Mömke and Svensson's circulation problem.

Lemma 5.2.2. *The support graph of the circulation is a 2-vertex-connected spanning subgraph in G.*

Proof. The demand assignment defined in 5.1 is the same assignment as in equation (3.7). For the capacity assignment, we know for the back edges, we would never set the flow to 2 because we can always cover the tree edge with flow 1, then we know the capacity assignment are the same. So we know there exists a 2-edge-connected spanning subgraph in D. We call the original nodes in G the $''$old nodes$''$, the new added nodes in D the $''$new nodes$''$. We know there is no edge coming into old nodes in D, so the final 2-edge-connected spanning subgraph must have some edge to cover the old nodes. After we transfer back to the original graph, for every node in original graph, there is at least one back edge through the tree cut across this node. Thus the support graph in D is a 2-edge-connected spanning subgraph, and is a 2-vertex-connected spanning subgraph in G.

So Lemma 5.2.2 is proved. □

Next we will prove the theorem in Mömke Svensson's paper [MS11]:

Theorem 5.2.3. *Given a 2-vertex-connected graph $G = (V, E)$, and a removable pairing (R, P) obtained from the depth first search tree T, if we can get a minimum cost circulation with cost $c(C)$, then we can find an Eulerian graph with the number of edges less than or equal to $\frac{4}{3}n + \frac{2}{3}c(C) - \frac{2}{3}$.*

Proof. Since our removable pairing is obtained from the depth first search tree, so for every tree edge t_v directed out from node v, at most one back edge can pair it, thus the rest of back edges coming into v will stay in set $R \setminus P'$ where P' denotes all edges in P. Note that there is a special case for root r: if there is only one incoming back edge towards the root, the back edge cannot be paired with the tree edge directed

out from r since every pair must incident to a node of degree at least 3. So

$$|R| - 2|P| = \begin{cases} c(C) \text{ if the root has more than one incoming back edge,} \\ c(C) + 1 \text{ otherwise.} \end{cases} \tag{5.3}$$

From Theorem 5.1.2, we know we can always get an Eulerian graph with edges less than or equal to $\frac{4}{3}|E| - \frac{2}{3}|R|$. Thus

$$\begin{aligned} \frac{4}{3}|E| - \frac{2}{3}|R| &= \frac{4}{3}(n - 1 + |R| - |P|) - \frac{2}{3}|R| \\ &= \frac{4}{3}n + \frac{2}{3}(|R| - 2|P|) - \frac{4}{3} \\ &= \begin{cases} \frac{4}{3}n + \frac{2}{3}c(C) - \frac{4}{3} \text{ if the root has more than one incoming back edge.} \\ \frac{4}{3}n + \frac{2}{3}c(C) - \frac{2}{3} \text{ otherwise.} \end{cases} \end{aligned}$$

From above analysis, we know we can always get an Eulerian graph with less than or equal to $\frac{4}{3}n + \frac{2}{3}c(C) - \frac{2}{3}$ edges.[2]

Hence Theorem 5.2.3 is proved. □

5.3 Removable Pairing Obtained by Ear Decomposition

Another way to get a removable pairing is from ear decomposition, as described in [SV12]. Given a 2-edge-connected graph G, if the graph has an ear decomposition without trivial ears, then we can find a removable pairing (R, P) as follows:

- For each non-pendant ear Q, select one node v inside Q that is the endpoint of another ear. Denote the two edges incident with v in Q as uv, wv, put these two edges in R, and make these two edges a pair in P.

[2]For any 2-vertex-connected graph, we can get a circulation with cost at most $6(1-\sqrt{2})n+(4\sqrt{2}-3)OPT_{SEP}$ where OPT_{SEP} denotes the optimal solution for SEP. See Appendix B for details.

- For each pendant ear Z, select one random edge inside Z and put it in R.

It can be easily seen that this satisfies all the conditions, thus is a valid removable pairing. Next we will prove the following theorem that appears in [SV12].

Theorem 5.3.1. *Given a 2-vertex-connected graph G and an ear decomposition without trivial ears, then we can get an Eulerian graph with the number of edges at most $\frac{4}{3}(|V|-1)+\frac{2}{3}\pi$ where π is the number of pendant ears.*

Proof. For any ear decomposition, the number of ears in this ear decomposition is $k = |E| - |V| + 1$. Then we define the removable pairing as above, and for every non-pendant ear we have two edges in R, and for every pendant ear, there is one edge in R, thus $|R| = 2k - \pi = 2(|E| - |V| + 1) - \pi$. From Theorem 5.1.2, we know there exists an Euler graph with the number of edges at most $\frac{4}{3}|E| - \frac{2}{3}|R| = \frac{4}{3}(|V|-1)+\frac{2}{3}\pi$. So Theorem 5.3.1 is proved. □

Chapter 6

$\frac{4}{3}$-proof for special $\frac{1}{3}$-integer vertex graph for graph TSP

Given any $\frac{1}{k}$-integer vertex x of the subtour elimination polytope, let $G_x = (V_x, E_x)$ be the support graph of x. If we replace each edge $e \in E_x$ by kx_e copies of e, and denote the new graph by $G = (V, E)$ with $V = V_x$, then for every $v \in V$, we have $\delta(v) = 2k$, and for every set $S \subset V,\ 2 \leq |S| \leq |E| - 2$, we have $|\delta(S)| \geq 2k$. Thus we know G is a $2k$-regular $2k$-edge-connected graph. If we have an r-approximation method for $2k$-regular $2k$-edge-connected graphs, then we have an r-approximation method for graph TSP in the support graph of $\frac{1}{k}$-integer vertex.

From Section 2.7 we know the lower bound for the subtour elimination problem is n for graph TSP, and the fractional 2-factor problem provides a lower bound for the subtour elimination problem. There is a conjecture which says that the worst case upper bound for the integrality gap ratio happens when the optimal solutions of these two problems are equal [SWvZ12], i.e. when the solution for the subtour elimination problem is n. Then suppose we could show the following:

6.0.1 For any graph TSP problem for which SEP has optimal value n, the integrality gap is $\frac{4}{3}$.

Then the truth of this conjecture would imply the integrality gap for graph TSP is $\frac{4}{3}$ in general, i.e. prove Conjecture 1.1.1 for graph TSP.

One approach we could use to show 6.0.1 is to show it is true for any graph TSP problem that comes from a graph G_x for a vertex x of P_{SEP}, i.e. for any $\frac{1}{k}$-integer vertex x. Note that 6.0.1 is already known to be true for $\frac{1}{2}$-integer x. In this chapter we examine the problem for $\frac{1}{3}$-integer x.

6.1 Two Proofs for The Result for Special Vertices

We begin with some notation for this section:
Let x be any $\frac{1}{3}$-integer vertex, and let $G_x = (V_x, E_x)$ be the support graph of x. If we triple all the 1-edges and double all the $\frac{2}{3}$-edges, we get a 6-regular 6-edge-connected graph $G = (V, E)$, and $V = V_x$. Since G_x comes from a $\frac{1}{3}$-integer vertex, then we can classify all the nodes into the following 7 categories (See Figure 6.1) [1]

Then we apply a similar algorithm as Algorithm 5.1. We only need to change Step 1 in the algorithm. When we build the depth first search tree, we select the edge with maximum x_e value. Note this is also done in [MS11] for their approximation algorithm for general graph TSP. Denote the depth first search tree we get from G to be T. We know there is a corresponding depth first search tree T_x in G_x. After we apply Mömke Svensson's Gadget, denote the new graph as $G' = (V', E')$.[2] Let T' be the tree edges in G', B and B' be the set of back edges in G and G' respectively. Then we define the circulation problem as described in Chapter 5, i.e. we set the

[1] To make our proof easier, we can shrink all the 1-paths to be 1-edges. So we do not have d_2 nodes here. Then if we can obtain a TSP tour in new graph with cost $\alpha n (\alpha \geq 1)$, then we can transfer back to a TSP tour less than or equal to αn because we only need to add one edge for one extra node.

[2] To make the notation easier, for any graph G and a depth first search tree T in G, if we apply Mömke Svensson's Gadget to G, then we denote the new graph to be G'. For example, the support graph G_x here would become G'_x and the 6-regular 6-edge-connected graph G would become G'.

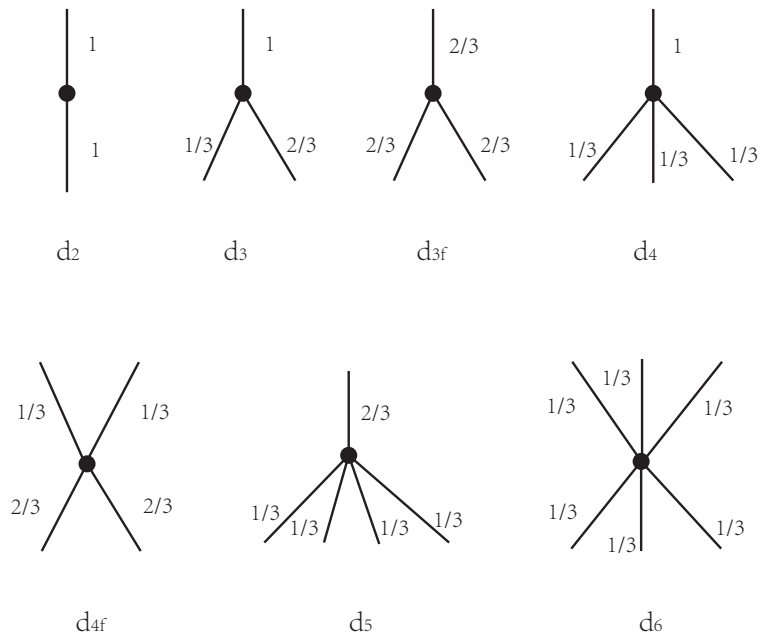


FIGURE 6.1: Seven categories of the nodes in G_x.

demands and capacities as follows:

$$d_e = \begin{cases} 1 \text{ for } e \in T', \\ 0 \text{ otherwise.} \end{cases} \qquad u_e = \begin{cases} 1 \text{ for } e \in B', \\ \infty \text{ otherwise.} \end{cases} \tag{6.1}$$

The cost will be the same as defined in 5.2. We know if there exist a feasible flow of cost $c(C)$, we can obtain a graph TSP solution with at most $n - 1 + c(C)$ edges.

Next we define the overflow nodes:

Given a graph and a directed tree in this graph, if we have a flow assignment f, then we call a node v to be an *overflow node* if:

$$\sum_{e \in B_v} f_e > 1, \text{ where } B_v \text{ is the set of incoming back edges towards } v.$$

We know from Chapter 5 that if the number of overflow nodes is 0, then we get a $\frac{4}{3}$-approximation algorithm.

In the following context, we will mainly deal with overflow nodes with a special

structure, denote the set of such nodes as O_{d_6}. A overflow node v is O_{d_6} if it satisfies the following two constraints:

- The incoming back edges towards v carry a total flow more than one.
- There are four edges with $x_e = \frac{1}{3}$ coming towards v.

From Figure 6.1, we can see in G_x only a d_6 node can be a O_{d_6} node, but in G all nodes are degree 6, see Figure 6.2 for example, all back edges in this figure carry $\frac{1}{3}$ flow, obviously not all degree 6 nodes are O_{d_6} nodes in this example.

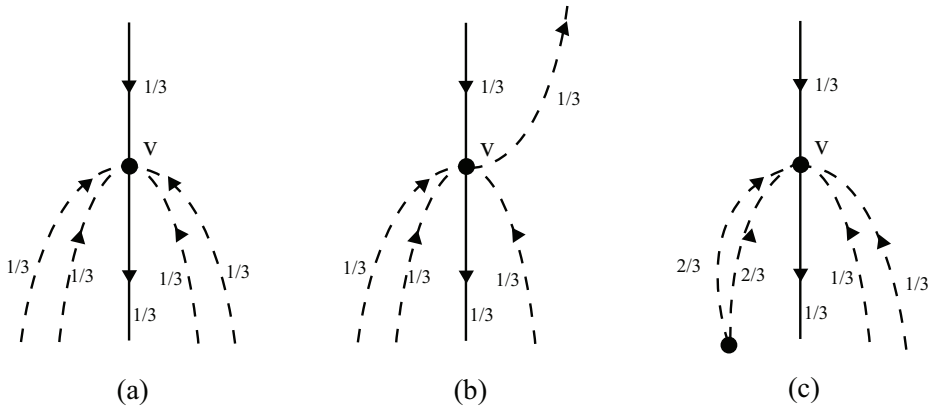


FIGURE 6.2: (a) An O_{d_6} node. (b) Not an O_{d_6} node since the incoming flow less than or equal to one. (c) Not an O_{d_6} node, since there are two back edges comes from an $\frac{2}{3}$-edge, not $\frac{1}{3}$-edge.

In this section we will use two methods to prove our result. The first proof will use the 6-regular 6-edge-connected graph G while the second proof will use the support graph G_x directly.

6.1.1 First Proof With $\frac{1}{3}, \frac{1}{4}$ Idea

We define the circulation problem as in (6.1), and use the total cost function as in (5.2). In other words, for the cost function defined in(5.2), every node v will receive one unit of flow at cost zero. Next we will prove the following theorem:

Theorem 6.1.1. *Given a graph G' constructed by the method above, we can show there exists an integer circulation in G' with cost at most $\frac{1}{3}|O_{d_6}|$. If G' doesn't contain an O_{d_6} node, then the cost of the circulation is 0.*

Proof. Denote the back edges in the tree cut through edge $e \in T$ in G by δ_{T_e}, and the tree cut through edge $e \in T'$ in G' to be δ'_{T_e}. Since G is 6-edge-connected, then for every tree edge $e \in T$, we have $|\delta_{T_e}| \geq 5$.
Then we manually set a flow f as follows[3]

$$f_e = \begin{cases} \frac{1}{4} \text{ for all } e \in B', \\ \sum |\frac{1}{4}\delta'_{T_e}| \text{ for all } e \in T'. \end{cases}$$

To show this is a feasible circulation, we have to prove $f_e \geq 1$ for every tree edge $e \in T'$. Let T_{w_i} be the subtree rooted at w_i in G, and T'_{w_i} be the subtree rooted at w_i in G'. According to the Mömke Svensson gadget, since we add one more node v_i to edge (v, w_i), then we have two kinds of tree edges:

1. For (v_i, w_i):

   Since all the back edges rooted in T_{w_i} to v will be redirected to v_i, so $\delta'_{T_{v_i w_i}}$ equals exactly the number of edges in $\delta_{T_{vw_i}}$. Because $|\delta_{T_{vw_i}}| \geq 5$, thus $f_{(v_i,w_i)} \geq \frac{5}{4}$.

2. For (v, v_i):

   If there exist an edge (v, v_i) with $f_{(v,v_i)} < 1$, i.e. less than 4 back edges directed from T'_{w_i} to the ancestor of v, i.e. $|\delta'_{T_{vv_i}}| < 4$. We consider 4 possible cases.

   - $|\delta'_{T_{vv_i}}| = 0$.

     That will mean that v was a cut vertex in G, thus this case would not happen.

[3]The idea derived from [R11], he applies the idea to k-regular k-edge-connected graph for graph TSP when k=4,5.

- $|\delta'_{T_{vv_i}}| = 1$.

  Let S be the node set including $V(T_{W_i})$ and v. Because $|\delta_{vw_i}| \geq 5$, and v is a degree 6 node, so the edges between v and T_{w_i} in G must be greater than or equal to 5, then the edges between v_i and T_{w_i} must be greater than or equal to 5. Figure 6.3 shows the structure satisfying this condition. However, $|\delta(S)| = 2$ violates 6-edge-connectivity, thus this case would not happen.

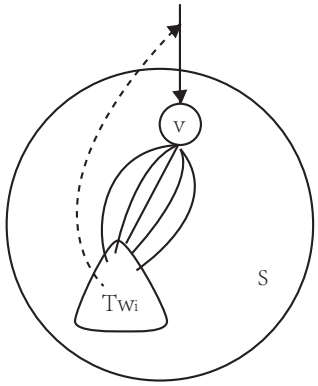


FIGURE 6.3: Graph which shows the situation in G when there is only one back edge rooted at T'_{w_i} to the ancestor of v.

- $|\delta'_{T_{vv_i}}| = 2$.

  This means there are two back edges rooted from T'_{w_i} to the ancestor of v. We can easily see that the number of edges between v and T_{w_i} is 4, so v has one more extra edge: it can be either a tree edge to another branch or a back edge to the ancestors of v (see Figure 6.4). But under both scenarios, we can see $|\delta(S)| < 6$, which contradicts 6-edge-connectivity. So this case would not happen.

- $|\delta'_{T_{vv_i}}| = 3$.

  We can do the same analysis, and get the three possible situations shown in Figure 6.5, but in all three situations in Figure 6.5 we have $|\delta(S)| = 6$, which satisfies 6-edge-connectivity. So under this scenario we also have

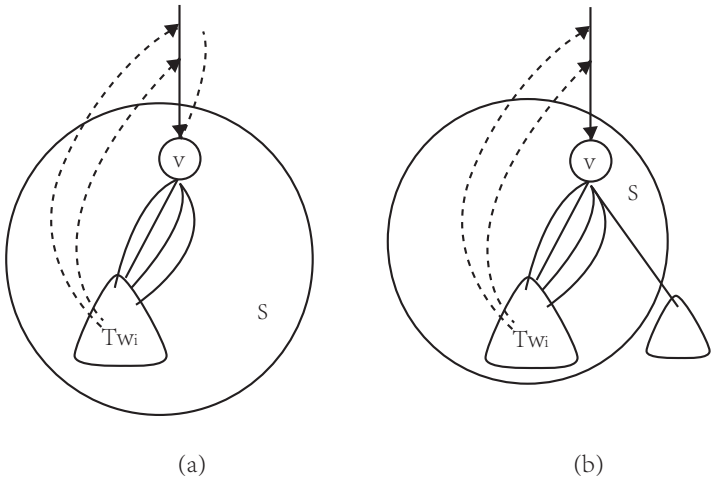


FIGURE 6.4: When there are two back edges rooted at T'_{w_i} to the ancestor of v, (a) and (b) shows the possible two cases that can happen in G.

$f_{vv_i} < 1$.

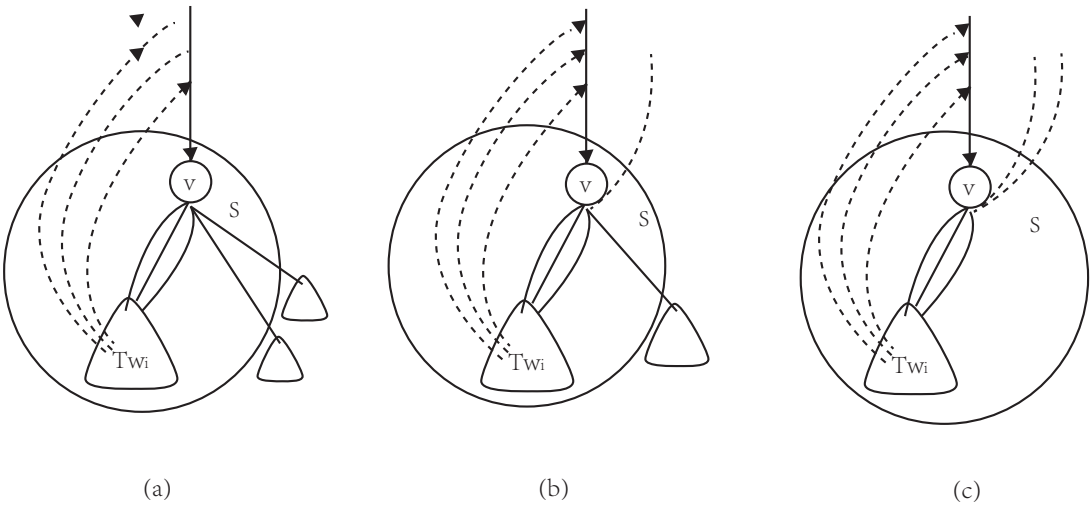


FIGURE 6.5: When there are three back edges rooted at T'_{w_i} to v, all corresponding graphs in G satisfy 6-edge-connectivity.

From the above analysis, we can see the problem here can be described as follows. To make this flow work, we assign f_e to be $\frac{1}{3}$ for all e in $\delta'_{T_{vv_i}}$. Because we always select the edge with maximum x_e to stay in the tree, so all the 1-edges stay in the tree. That means some $\frac{1}{3}$-edges and $\frac{2}{3}$-edges would hold the flow of $\frac{1}{3}$. Since only the root and newly added node v_i have incoming back edges, and for every node $v_i \in V'$, there are at most four back edges coming into v_i, therefore there are four scenarios

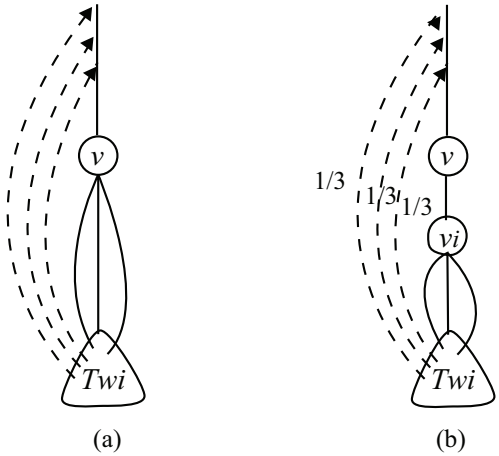


FIGURE 6.6: Situation where we need to assign $\frac{1}{3}$ on the back edges. (a) The situation in G. (b) The situation in G'.

below where the flow coming into v_i would exceed one:

(1) $\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}$. (2) $\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{4}$. (3) $\frac{1}{3}, \frac{1}{3}, \frac{1}{4}, \frac{1}{4}$. (4) $\frac{1}{3}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}$.

Claim 1: *All four back edges in scenario (1) can only be $\frac{1}{3}$-edges.*

Proof. Denote the tree edges starting from node v in T' to be t'_v. Since G comes from a vertex, we have to double the $\frac{2}{3}$-edge and triple the 1-edge. So in any cut, we would have two parallel $\frac{2}{3}$-edges or three parallel 1-edges. Assume there exist a 1-edge or $\frac{2}{3}$-edge among these four edges, and build the depth first search tree accordingly. We always select the edge with maximum x_e, so that t'_v must be a 1-edge or $\frac{2}{3}$-edge, and there exists a parallel 1-edge or $\frac{2}{3}$-edge e directed into v. It can be easily seen from Figure 6.6 that the back edge parallel to the tree edge would not be assigned $\frac{1}{3}$, thus $f_e = \frac{1}{4}$. This contradicts the fact that all four back edges carry flow $\frac{1}{3}$. □

So we know if all four back edges are $\frac{1}{3}$-edges, it is definitely a O_{d_6} node.

Subsequently we have the following two scenarios:

- **CASE 1**: We do not have O_{d_6} nodes.

  From Claim 1 we know we only have situation (2) to (4) now, so there would be

at least one 1-edge or $\frac{2}{3}$-edge among these four edges, so we have the following two situations:

1. The 1-edge or $\frac{2}{3}$-edge are parallel back edges. Because we always select the edge with maximum x_e value, t'_{v_i} must be another 1-edge or $\frac{2}{3}$-edge. It can be easily seen that the cut through tree edge e is not the same as in Figure 6.6 because there are only two back edges towards v, so we know the parallel edge e' to e would carry flow $\frac{1}{4}$, and this edge only covers the tree edge t'_{v_i}. Because we increase the flow from $\frac{1}{4}$ to $\frac{1}{3}$, so we can decrease the flow on e' to make sure the flow coming into v is less than or equal to one. If we decrease e' from $\frac{1}{4}$ to 0, that is equivalent to having only three back edges directed to v. If each of the three edges carries flow $\frac{1}{3}$, we can see the total flow coming into v_i equals one. Next we need to show this can still maintain a feasible flow. Since we only decrease flow on e', it affects the flow on e. Since $|\delta'_{T_e}| \geq 5$, and we only have four back edges present, so we know there must exist another back edge carrying a flow of at least $\frac{1}{4}$ from T_{w_i} to v_i. So we have a flow of at least $\frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{3} > 1$, thus this is still a feasible flow. This implies there exists a feasible circulation with cost zero.

2. If the 1-edge or $\frac{2}{3}$-edge is a back edge parallel to a tree edge. We can do the same analysis, and it can be easily seen that this back edge would carry flow $\frac{1}{4}$, so that we can still decrease the flow and get a zero cost circulation.

Thus from above analysis, we know if there is no O_{d_6} node, then we would get a feasible circulation with cost zero, i.e. for every node v, there would be at most one incoming back edge.

- **CASE 2**: We have O_{d_6} nodes.

  A O_{d_6} node would have at most four incoming back edges with flow $\frac{1}{3}$. Thus for

every O_{d_6} node, the cost would be at most $\frac{1}{3}$. Therefore if O_{d_6} node are present, a feasible circulation would have a cost of at most $\frac{1}{3}|O_{d_6}|$.

Hence Theorem 6.1.1 is proved. □

Theorem 5.2.2 shows that given a circulation with cost $c(C)$ in G', a 2-vertex-connected graph with edges less than or equal to $n - 1 + c(C)$ can be obtained.

Theorem 6.1.2. *Given a $\frac{1}{3}$-integer vertex x, and the support graph G_x of x, let $G = (V, E)$ be the 6-regular 6-edge-connected graph obtained from G, let T be the depth first search tree in G, and let O_{d_6} be the special overflow nodes in T. We can find a TSP tour with the number of edges less than or equal to $\frac{4}{3}n + \frac{2}{9}|O_{d_6}| - \frac{2}{3}$. Moreover, if we don't have O_{d_6} nodes, we can get a result of $\frac{4}{3}n$.*

Proof. From Theorem 6.1.1, we know we can get a minimum cost circulation with cost $c(C) \leq \frac{1}{3}|O_{d_6}|$. From Theorem 5.2.3, we know we can find an Eulerian graph with the number of edges less than or equal to $\frac{4}{3}n + \frac{2}{3}c(C) - \frac{2}{3} = \frac{4}{3}n + \frac{2}{9}|O_{d_6}| - \frac{2}{3}$. From Theorem 2.8.1, we know we can also find a TSP tour with the number of edges less than or equal to $\frac{4}{3}n + \frac{2}{9}|O_{d_6}| - \frac{2}{3}$. So Theorem 6.1.2 is proved. □

Next we use an example to illustrate this idea:

Example Given the support graph G_x (see Figure 6.7(a)) of a $\frac{1}{3}$-integer vertex, double all the $\frac{2}{3}$-edge and triple all the 1-edge to obtain a 6-regular 6-edge-connected graph G (see Figure 6.7(b)). Let node A be the root, build a depth first search tree T starting from A (see Figure 6.8 (c)), with each iteration selecting the maximum x_e. Then apply Mömke Svensson's Gadget to G to get the graph G'. If we set every back edge with flow $\frac{1}{4}$, then the cut shown in Figure 6.9(e) would have flow less than one, and we have to increase some flow to $\frac{1}{3}$ (see Figure 6.9(f)). Thus a minimum cost circulation can be obtained as in Figure 6.10(g), and a minimum cost odd join can be found as shown in Figure 6.10(h). Then we get an Eulerian graph by $H + J \setminus R - J \cap R$ (see Figure 6.11(i)), in this case, the Eulerian graph is also a TSP tour.

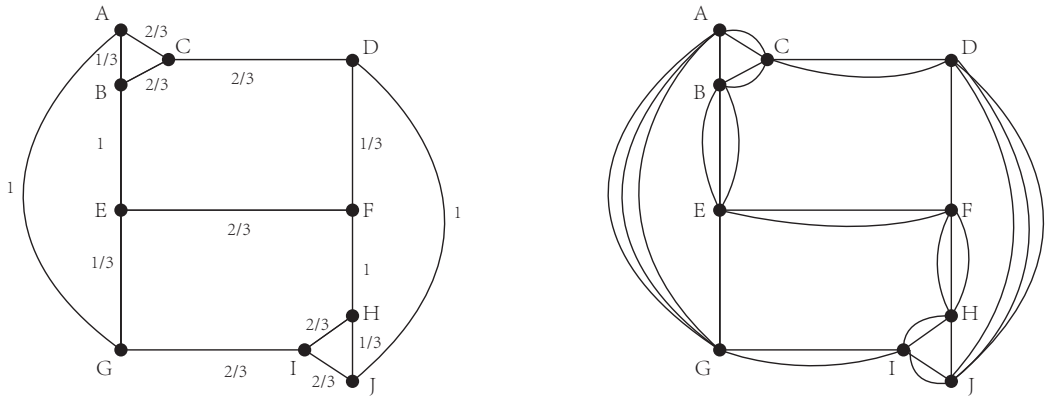


FIGURE 6.7: (a) Support graph G_x of a $\frac{1}{3}$-integer vertex. (b) The 6-regular 6-edge-connected G graph from G_x.

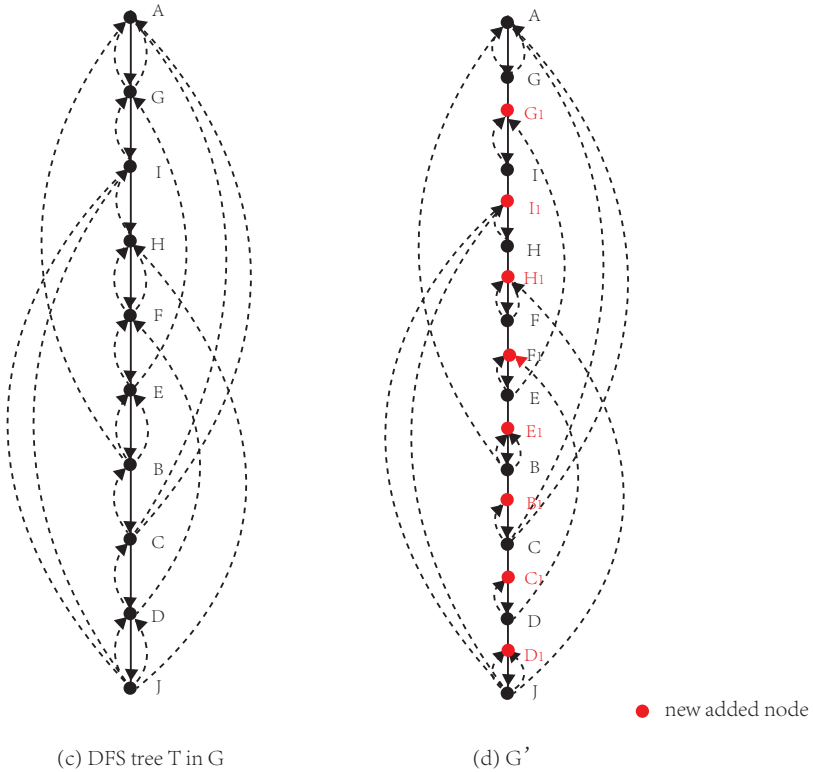


FIGURE 6.8: (c) Depth First Search Tree T in G. (d) Apply Mömke Svensson's Gadget to G, get graph G'.

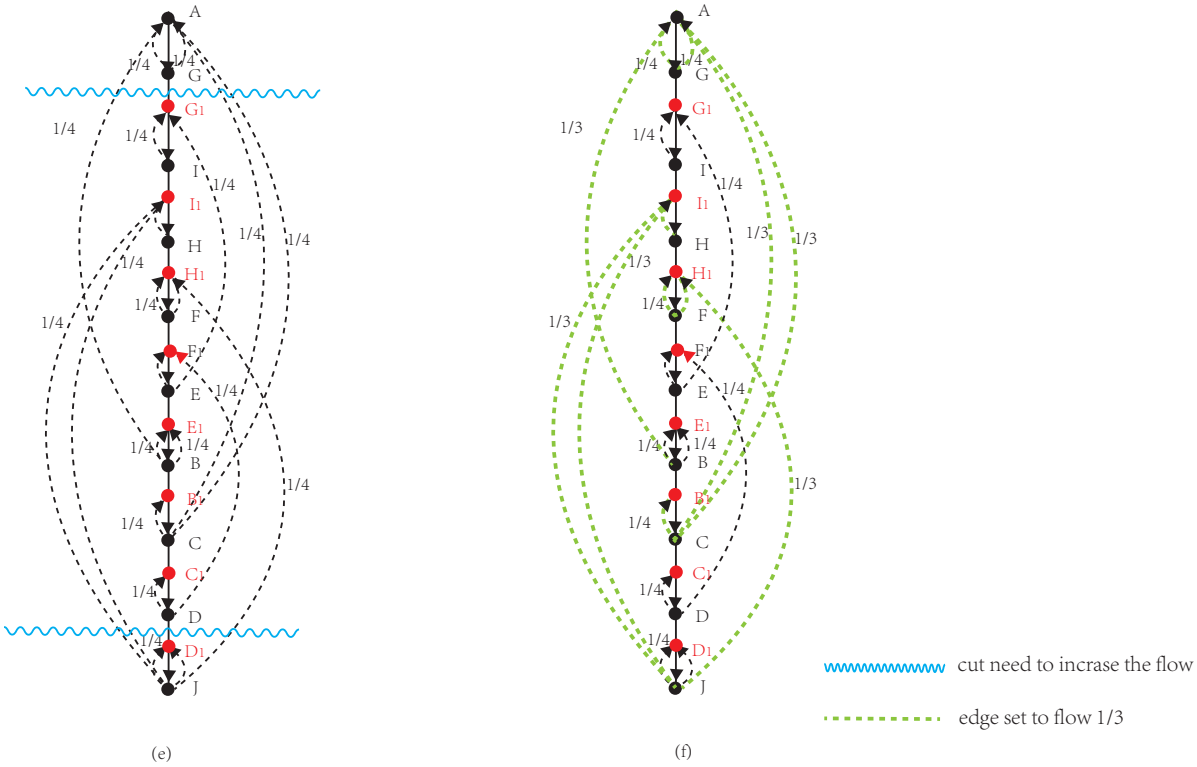


FIGURE 6.9: (e) Set every back edge with flow $\frac{1}{4}$. (f) Increase some flow to $\frac{1}{3}$.

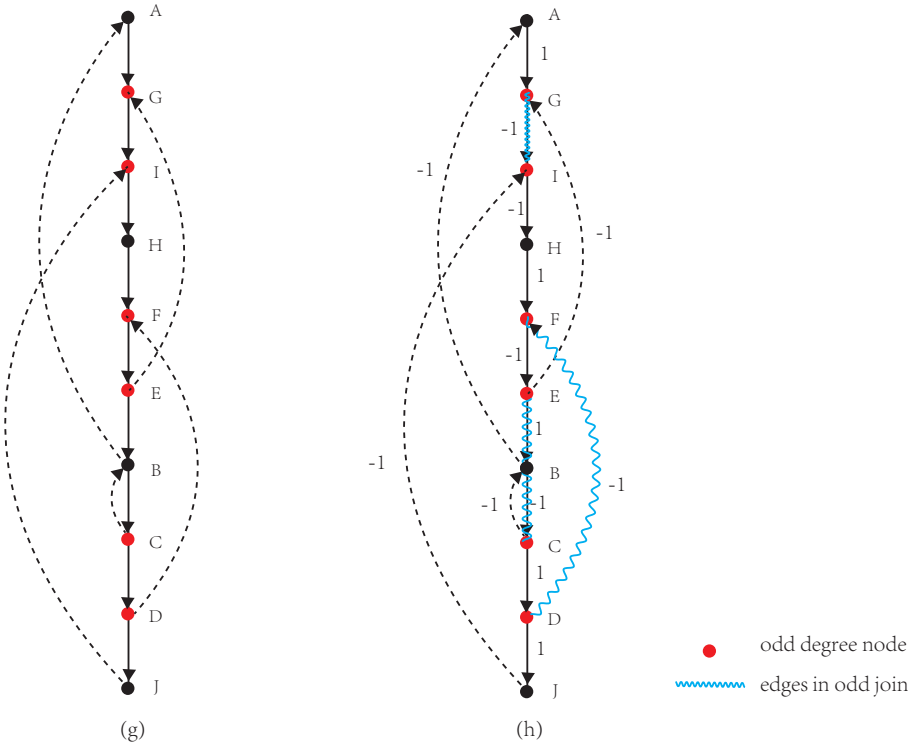
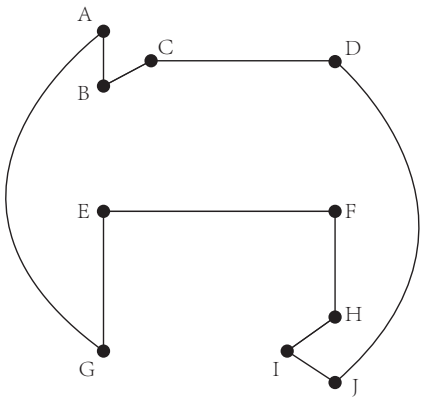


FIGURE 6.10: (g)The support graph H of a minimum cost circulation. (h) Minimum cost odd join J in H.



(i) Eulerian graph

FIGURE 6.11: (i)Eulerian graph obtained from $H + J \setminus R - J \cap R$.

6.1.2 Second Proof With x_e Value Idea

The second proof uses the support graph G_x of x, and makes a feasible circulation with the x_e value as the flow on each back edge. As mentioned in the beginning of this section, let G'_x be the graph G_x with Mömke Svensson's gadget, T_x and T'_x be the depth first search tree in G_x and G'_x, B and B' be the set of back edges in G_x and G'_x. Let δ_{T_e} and $\delta'_{T_{e'}}$ be the set of back edges in tree cut through $e \in T_x$ and $e' \in T'_x$. We prove a theorem similar to Theorem 6.1.1:

Theorem 6.1.3. *Given the support graph G_x from a $\frac{1}{3}$-integer vertex x, and the graph G'_x from G_x with the Mömke Svensson Gadget, then there exists an integer circulation in G'_x with cost at most $\frac{1}{3}|O_{d_6}|$.*

Proof. First we will manually set a feasible flow in G'_x,

$$f_e = \begin{cases} x_e \text{ for all } e \in B', \\ \sum_{e \in \delta'_{T_e}} x_e \text{ otherwise.} \end{cases} \tag{6.2}$$

From Theorem 6.1.1 we know if we want to show this is a feasible circulation, we have to show $f_e \geq 1$ for all $e = (v, v_i)$ or (v_i, w_i).

1. For (v_i, w_i):

We know all edges directed into v will be directed to v_i, so the edges in $\delta'_{T_{v_iw_i}}$ carry exactly the same x_e value as in $\delta_{T_{vw_i}}$, i.e. $\sum_{e\in\delta'_{T_{v_iw_i}}} x_e = \sum_{e\in\delta_{T_{vv_i}}} x_e$. From the subtour elimination constraint we know for every cut we have $\sum_{e\in\delta(S)} x_e \geq 2$, so all the tree cuts must have $\sum_{e\in\delta_{T_{vw_i}}} x_e + x_{vw_i} \geq 2$. We know for any $e \in E_x$ in a $\frac{1}{3}$-integer vertex, maximum value of x_e is 1. Thus for any tree cut, we have $\sum_{e\in\delta_{T_{vw_i}}} x_e \geq 1$. Because for every back edge $e \in B'$, the f_e value is the same as x_e value, then for any v_iw_i, we have $\sum_{e\in\delta_{T'_{vw_i}}} x_e \geq 1$, thus $f_{v_iw_i} \geq 1$.

2. For vv_i:

   We know if we delete any node v in tree T_x, the tree will break into two parts T_{x1} and T_{x2}. We call the edges between $V(T_{x1})$ and $V(T_{x2})$ a *node cut*, denoted as δ_v.

   We begin by proving the following claim:

   Claim 2: *For any $v \in V_x (v \neq r)$, $\sum_{e\in\delta_v} x_e \geq 1$.*

   Proof. Let e_1 and e_2 be the two tree edges incident to v, from the subtour elimination constraint, we know

$$\begin{aligned} (1)\ x_{e_1} + \sum_{e\in\delta_v} x_e &\geq 2. \\ (2)\ x_{e_2} + \sum_{e\in\delta_v} x_e &\geq 2. \end{aligned}$$

   Also, from the degree constraint, we have $x_{e1} + x_{e2} = 2$. Then we sum (1) and (2), and get $x_{e_1} + x_{e_2} + 2\sum_{e\in\delta_v} x_e \geq 4$, thus $\sum_{e\in\delta_v} x_e \geq 1$. So Claim 2 is proved. □

   We can see that the edges in $\delta'_{T_{vv_i}}$ are identical to the edges in δ_v, thus $f_{vv_i} \geq 1$.

From the above two situations, we can see the assignment in (6.2) is a feasible circulation.

To analyze the cost of this circulation, refer to Figure 6.3. We know there are 7 kinds of nodes in G_x. Because we always select the edge with maximum x_e to stay in the tree, from Figure 6.12, we know only d_6 nodes would have the sum of x_e value of back edges exceeding one. So all those nodes are O_{d_6} nodes in G_x. We know all

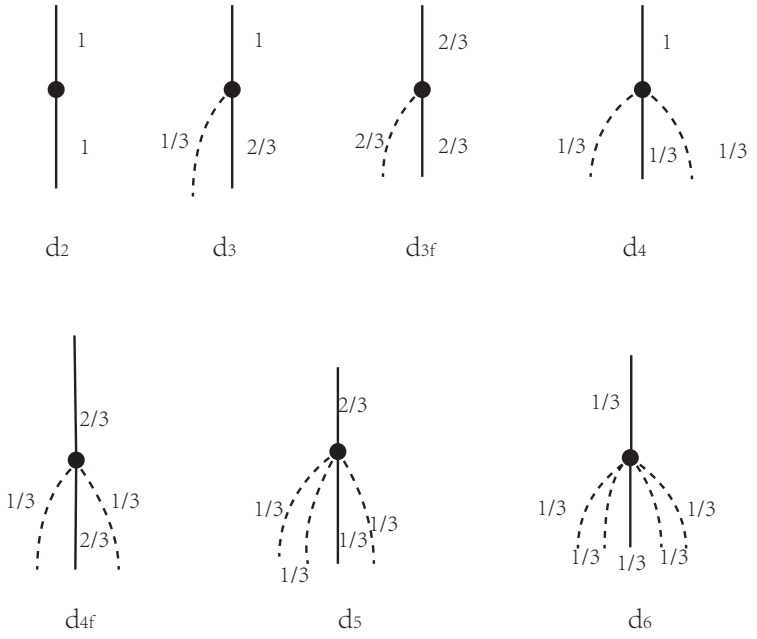


FIGURE 6.12: If we select maximum x_e value to stay in the depth first search tree, then we have the above seven situations in G_x.

back edges directed to v will be redirected to v_i, thus the flow coming into v_i in G'_x equals the flow coming into v in G_x. And for every O_{d_6} node, we have an extra $\frac{1}{3}$ flow beyond 1, so the cost of the circulation is at most $\frac{1}{3}|O_{d_6}|$. From Theorem 3.1.3, we know there exists an integer circulation with cost less than or equal to $\frac{1}{3}|O_{d_6}|$.

Hence Theorem 6.1.3 is proved. □

Combining the result in Theorem 5.2.3 and Theorem 2.8.1, we can find a TSP tour with edges less than or equal to $\frac{4}{3}n + \frac{2}{9}|O_{d_6}| - \frac{2}{3}$.

Although we get the same result from both proofs, the numbers of O_{d_6} nodes in the two situations are not the same. In the second method, if a d_6 has four incoming

back edges, then the flow would exceed one. While in the first method, if all four back edges carry flow $\frac{1}{4}$, then this d_6 node would not become a O_{d_6} node.

6.2 Analysis Of $\frac{1}{3}$-Integer Vertex and O_{d_6} node

As in Boyd and Pulleyblank's paper [BP91], given any vertex x in polytope of subtour elimination problem, the number of edges in support graph G_x of x is at most $2n - \frac{k}{2} - 3$, where k denotes the number of nodes incident with three edges, and none of them has x_e value equal to one. Then in our case, for a $\frac{1}{3}$-integer vertex, we know the number of edges is at most $2n - \frac{|d_{3f}|}{2} - 3$, [4] i.e.

$$\frac{2|d_2| + 3(|d_3| + |d_{3f}|) + 4(|d_4| + |d_{4f}|) + 5|d_5| + 6|d_6|}{2} \leq 2n - \frac{|d_{3f}|}{2} - 3. \tag{6.3}$$

Also, since all nodes would fall into any of those seven categories, then

$$|d_2| + |d_3| + |d_{3f}| + |d_4| + |d_{4f}| + |d_5| + |d_6| = n. \tag{6.4}$$

Combining equation (6.2) and (6.4), we have

$$|d_6| \leq \frac{|d_3| - |d_5| - 6}{2}. \tag{6.5}$$

So the next question would be: If we have a d_6 node, is it possible that node would never become a O_{d_6} node? Unfortunately, the answer is no, see the example below:

Example Figure 6.13 shows the support graph G of a $\frac{1}{3}$-integer vertex with 16 nodes. Then we build the depth first search tree starting from node 10 (see Figure 6.14). It can be easily seen from this graph that no matter which graph we are dealing with, we have to assign $\frac{1}{3}$ to the edges in the green cut, thus node 9 would become a O_{d_6}

[4] See Figure 6.1 for the structure of d_{3f} node, i.e. the degree 3 node not incident with a 1-edge.

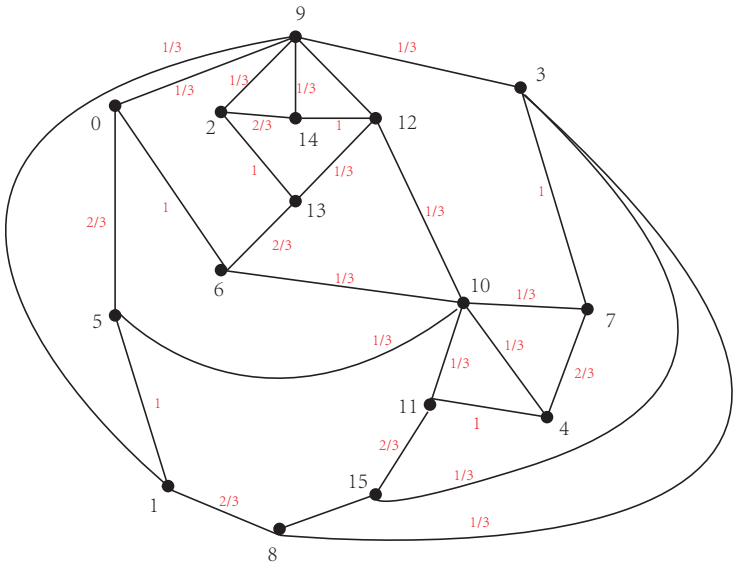


FIGURE 6.13: Support graph G of an $\frac{1}{3}$-integer vertex.

node[5].

For any 6-regular 6-edge-connected graph, the best result for graph TSP is $\frac{7}{5}n$ from Sebő's paper [SV12]. From Equation (6.5), we know if $n \leq 9$, $|d_6| \leq 1$, thus when we build the depth first search tree, we can start from a node $v, v \notin d_6$, thus we have the cost of the fractional circulation of $\frac{1}{3} + \frac{1}{3} = \frac{2}{3}$.[6] Because we know we can get an integer circulation with cost less than or equal to $\frac{2}{3}$, and since the flow on every edge is an integer, therefore the final cost must be integer, i.e. we can get an integer circulation with cost zero. Then we can get an Eulerian graph with edges less than or equal to $\frac{4}{3}n$ when $n \leq 9$.

Because the support graph from a vertex has some special properties, then the next question for us would be: Can we have many d_6 nodes in the support graph of a vertex? Using the list of all non-isomorphic vertices with less than or equal to 12 nodes[7], we designed a program using C++ under the Linux system to find all the

[5]Note that we can manually convert from a vertex to another vertex, so if we have a support graph of a vertex with d_6 node, then we can get a graph of another vertex with more O_{d_6} nodes, see Appendix A for details.

[6]The first $\frac{1}{3}$ comes from a O_{d_6} node, because there is only one d_6 node in graph, thus the maximum back edges coming into root is 4, thus the root has extra flow of at most $\frac{1}{3}$.

[7]A list of vertices from the subtour elimination polytope can be found at

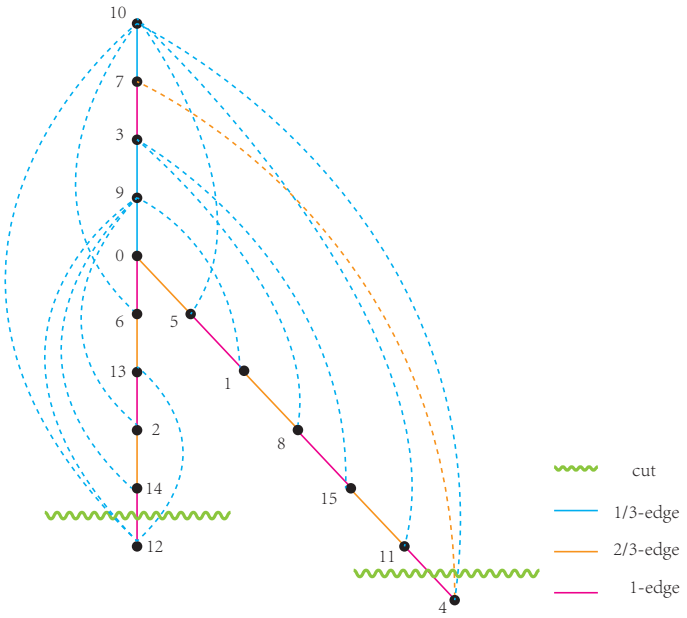


FIGURE 6.14: Depth first search tree that causes problems.

$\frac{1}{3}$-vertices with degree 6 nodes. From the results of our experiments, we can see the number of support graphs with d_6 nodes when $10 \leq n \leq 12$, as shown in Table 6.1. Note that all of these graphs contain only one d_6 node. From the table, we can also see only a small set of graphs have d_6 nodes.

TABLE 6.1: Number of graph with d_6 nodes when $10 \leq n \leq 12$.

	10	11	12
Number of graphs with d_6 nodes	6	10	40
Number of non-isomorphic $\frac{1}{3}$-vertices	204	2738	35589
Percentage of non-isomorphic $\frac{1}{3}$-vertices with d_6 nodes	0.0294	0.0036	0.0011
Total number of non-isomorphic vertices	461	4972	68320

In general, since the number of edges is less than or equal to $2n - \frac{|d_{3f}|}{2} - 3$, then the number of back edges $|B| \leq n - 2 - \frac{|d_{3f}|}{2}$. Because we know every O_{d_6} node contains four back edges, so the number of O_{d_6} nodes is bounded as follows:

$$|O_{d_6}| \leq \frac{n-2}{4} - \frac{|d_{3f}|}{8}.$$

http://www.site.uottawa.ca/~sylvia/subtourvertices/index.htm.

From Theorem 6.1.2, we know the number of edges is at most

$$\begin{aligned} \frac{4}{3}n + \frac{2}{9}|O_{d_6}| - \frac{2}{3} &\leq \frac{4}{3}n + \frac{2}{9}(\frac{n-2}{4} - \frac{|d_{3f}|}{8}) - \frac{2}{3} && (6.6) \\ &= \frac{25}{18}n - \frac{7}{9} - \frac{|d_{3f}|}{36} && (6.7) \\ &< \frac{25}{18}n. && (6.8) \end{aligned}$$

Thus we know we can get a TSP tour with the number of edges $\leq \frac{4}{3}n$ if we do not have O_{d_6} nodes, and otherwise a tour with the number of edges less than $\frac{7}{5}n$, which is the current best approximation ratio currently known for graph TSP.

Chapter 7

Conclusion

In this thesis we have focused on approximation algorithms for 2ECSS and graph TSP. We present an approximation algorithm for 2ECSS that ensures the number of edges is at most $n+\frac{m-n}{k-1}-\frac{k-2}{k-1}$ for any k-edge-connected graph using circulations. The proof is much easier compared to the one presented by Huh [Huh04] for an approximation of $n+\frac{m-n}{k-1}$. Then we investigate some special vectors and show they are in the T-join polytope, which could be useful in future approximation algorithms. We give a clearer explanation of some of the results stated in [MS11] by combining the results found in [MS11],[SV12] and [BSvdSS11]. Finally, we present an approximation algorithm for the support graph of the $\frac{1}{3}$-integer vertices with ratio less than $\frac{7}{5}$, which is the optimal bound currently known for such graphs. Moreover, if we do not have O_{d_6} nodes, our algorithm ensures a ratio of $\frac{4}{3}$.

With the results we have, there are many directions for future work:

- For 2ECSS, using our new circulation method, we only get a $\frac{4}{3}$-approximation for k-regular, k-edge-connected graphs where $k \leq 4$, but in [SV12], they provide an algorithm for any 2-edge-connected graph that ensures the ratio of $\frac{4}{3}$. So it would be interesting to investigate ways to improve on the circulation method to get a better result.

- For graph TSP, although we obtain an algorithm which improves a little ($\frac{1}{18}$) on the approximation ratio of $\frac{7}{5}$ provided in [SV12], the algorithms only focus on the graphs that come from a $\frac{1}{3}$-vertex which can also be related to a 6-regular 6-edge-connected graph. The next step for us would be to generalize the idea and apply it to the $\frac{1}{k}$-vertex. In doing so we could possibly obtain an improvement for graph TSP wherever the optimal solution for SEP is n.

- With the study of T-joins, we give some special vectors that are in the T-join polytope, but we did not apply any of them to graph TSP. The next step for us would be to try to use these vectors for special sets of graphs, and achieve better results for graph TSP.

- With the computational study of vertices with O_{d_6} nodes, we didn't get too much information about the structure of the support graphs of such vertices. We could study more about the structure and learn more properties of those graphs which may be useful in algorithms.

- For the definition of removable pairing from ear decomposition, we know Sebő et al. [SV12] only get one removable edge for every pendant ear, the question for us is: can we find a way to get two removable edges for every ear, thus achieve the $\frac{4}{3}$ result? Another way to study this is to find a removable pairing from other structures.

Appendix A

Gadget To Generate Degree 6 Nodes

From Boyd and Elliott-Magwood's [BEM07] paper, we know the following theorem:

Theorem A.0.1. *Given a vertex x in P_{SEP}, let the support graph of x be $G = (V, E)$. If there is an edge $zw \in E$, and we can partition the edges in G incident with z into two parts $E_{z1},\ E_{z2}$ such that $0 \leq x(E_{z1}),\ x(E_{z2}) \leq 1$, then create the new graph G' as in Figure 1.1, and the vector created by G' is also a vertex in P_{SEP}.*

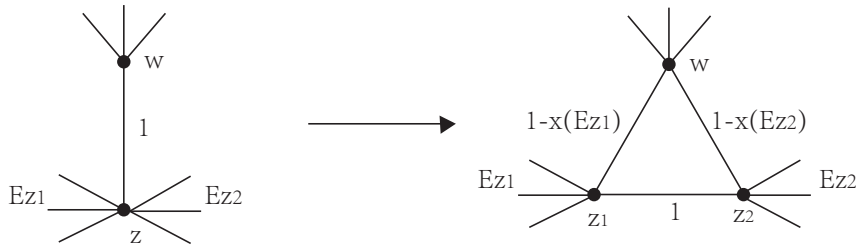


FIGURE 1.1: Split the edge to get a new vertex.

Next we will show how we can use the above theorem to transfer a $\frac{1}{3}$-integer vertex to another $\frac{1}{3}$-integer vertex with more degree 6 nodes.

Given the support graph G of any $\frac{1}{3}$-integer vertex with a d_4 node[1], w in the graph

[1]See Figure 6.1 for categories of nodes.

(see Figure 1.2(a)), then we can apply Theorem A.0.1 to edge wz, and we get the graph shown in Figure 1.2(b). We know z_2 must be a d_3 node, thus this means E_{z2} only consists of a $\frac{1}{3}$-edge. This can be done since any node incident with a 1-edge must also be incident with a $\frac{1}{3}$-edge[2]. Then we continue splitting edge z_1z_2, getting

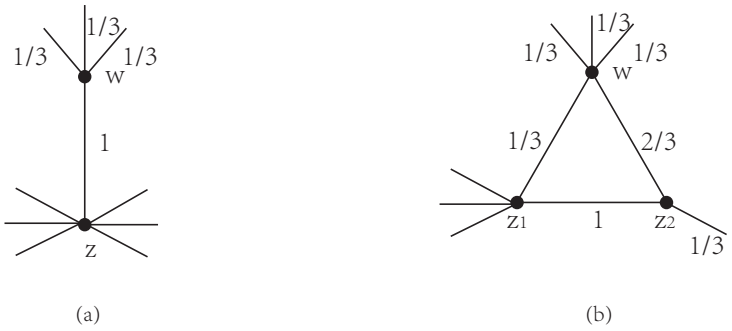


FIGURE 1.2: (a) The support graph G of a $\frac{1}{3}$-integer vertex. (b) The graph we get after we split edge wz.

the graph shown in Figure 1.3(c), and we can see that z_2 is a d_{4f} node (i.e. degree 4 node with no 1-edges incident). Finally we split edge wz_2, and get the graph shown in Figure 1.3(d). We can see that w is a d_6 node. Moreover, z_{22} is a d_4 node, thus we

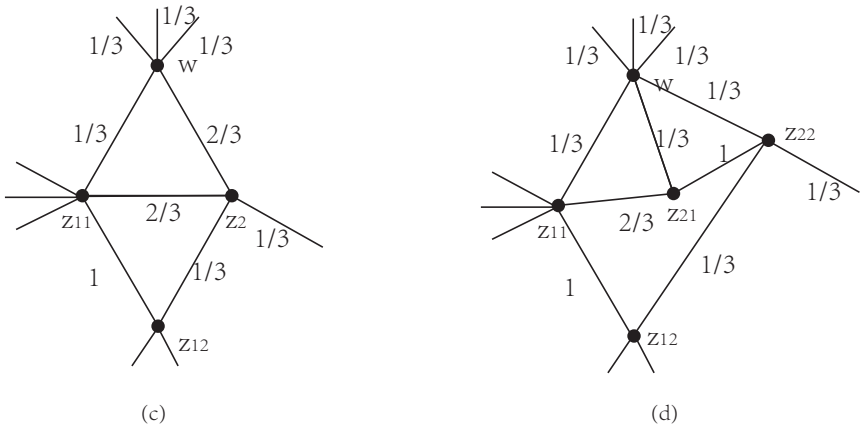


FIGURE 1.3: (c) Result graph after we split edge z_1z_2. (d) Final graph with d_6 node after we split wz_2.

can apply the same operations to get more d_6 nodes.

[2]Because we shrink all the 1-paths to 1-edges, so we do not have a d_2 node.

Appendix B

Details Of Mömke Svensson's Idea for Circulations

In this appendix we will prove the theorem stated in Mömke and Svensson's [MS11] paper.

Theorem B.0.2. *Given a 2-vertex-connected graph $G = (V, E)$ with at most $2n - 1$ edges*[1]*, we can get a 2-vertex-connected spanning subgraph using circulations. Moreover, the number of edges in this 2-vertex-connected spanning subgraph is $\leq 6(1 - \sqrt{2})n + (4\sqrt{2} - 3)OPT_{LP}(G)$ where $OPT_{LP}(G)$ denotes the optimal solution for the LP-relaxation of the graph TSP problem.*

Proof. Let x^* be the optimal solution for the LP relaxation of the TSP problem. Then they apply the following idea: First grow a depth first tree, picking the edge for each iteration with maximum x_e value.

From Lemma 5.2.2, we know the support graph of the circulation is a 2-vertex-connected spanning subgraph. Next we will prove the cost for the whole circulation $c(C)$ is $\leq 6(1 - \sqrt{2})n + (4\sqrt{2} - 3)OPT_{LP}(G)$.

[1]Because Mömke and Svensson deal with the support graph of optimal solutions of the LP for the TSP problem, we know for any extreme point for the polytope, the support graph has at most $2n - 1$ edges [GC85].

The main idea is to manually set a feasible circulation: the circulation f will in turn be the sum of two circulations f' and f'', and we obtain the two circulation as follows:

- f': for each back arc a we push a flow of size min[x_a^*,1] along the cycle formed by a and the tree-arcs in D, x_a^* denote the value of the corresponding edge in x^* of graph G.

- f'': for each node v of G that is replaced by a gadget consisting of an out-vertex v and a set Γ_v of in-vertices, we push for each $w \in \Gamma_v$ a flow of size max[1-$f'_{(v,w)}$,0] along the cycle that includes the arc(v, w) (and one back-arc).

Now we prove that f=f'+f'' defines a feasible circulation (i.e. for each edge a$\in Ts$, the circulation satisfies the demand $f_a \geq$1). From the construction of f'', we know

$$f' = \begin{cases} 1 \text{ when } x_a^* \geq 1, \\ x_a^* \text{ when } 0 \leq x_a^* \leq 1, \end{cases}$$

$$f'' = \begin{cases} 0 \text{ when } x_a^* \geq 1, \\ 1 - x_a^* \text{ when } 0 \leq x_a^* \leq 1, \end{cases}$$

so for all edges $a \in A$, we have f=f'+f'' equals 1, so we know this circulation is feasible.

Next we will analyse the cost of this circulation. The cost of f equals $cf = \sum_{v\in\Gamma} max[f(B(v)) - 1, 0]$, where Γ is the set of all in-vertices and $B(v)$ is the set of

all incoming back-arcs of $v \in \Gamma$. so the cost is upper-bounded as follows:

$$\begin{aligned} c(f) &\leq \sum_{v\in\Gamma}(max[f'(B(v))-1,0]+max[f''(B(v))-1,0]) \\ &\leq \sum_{v\in\Gamma}(max[f'(B(v))-1,0]+f''(B(v))). \end{aligned}$$

Then they analyse these two costs separately, and obtain the following:

$$\begin{aligned} &(1)\quad \sum_{v\in\Gamma} max[f'(B(v))-1,0] \leq (7-6\sqrt{2})n+4(\sqrt{2}-1)OPT_{LP}(G), \text{ and} \\ &(2)\quad \sum_{v\in\Gamma} f''(B(v)) \leq OPT_{LP}(G)-n. \end{aligned}$$

- For (1): Because G has at most $2n-1$ edges, the number of back-edges is at most n. $B(v)$ denotes the set of incoming back-edges of v, and maximum value $x^*_{t_v}$ among all $a \in B(v)$ where t_v is the outgoing tree arc of $v \in \Gamma$. Since in each iteration we choose the maximum x^*_a, so $x^*_a \leq x^*_{t_v}$. Furthermore, from the definition, f'_a=min[x^*_a,1] for each back-edge. So the number of back-edges in $B(v)$ is at least $\lceil \frac{f'(B(v))}{min[x^*(t_v),1]} \rceil$. Combining with the first argument above, we obtain the following inequality:

$$\lceil \frac{f'(B(v))}{min[x^*(t_v),1]} \rceil \leq n. \tag{B.1}$$

  For $v \in \Gamma$, they partition the flow into $l_v = \min[2-x^*(t_v), f'(B(v))]$ and $u_v = f'(B(v)) - l_v$. This partition is a valid partition because $l_v + u_v = f'(B(v))$ always holds.

  Because

$$\begin{aligned} \lceil \frac{f'(B(v))}{min[x^*(t_v),1]} \rceil - u_v &= \lceil \frac{f'(B(v))}{min[x^*(t_v),1]} \rceil - (f'(B(v)) - l_v) \\ &= \lceil \frac{f'(B(v)) - min[x^*(t_v),1](f'(B(v)) - l_v)}{min[x^*(t_v),1]} \rceil \end{aligned}$$

So:

1. If $x^*(t_v) \geq 1$, the result from above equals l_v, thus is $\geq \frac{l_v}{x^*(t_v)}$.
2. If $x^*(t_v) \leq 1$, the result from above equals $l_v + \frac{f'(B(v))(1-x^*(t_v))}{x^*(t_v)} = \frac{(u_v(1-x^*(t_v))+l_v)}{x^*(t_v)}$, thus is $\geq \frac{l_v}{x^*(t_v)}$.

From the above analysis, we have

$$\lceil \frac{f'(B(v))}{min[x^*(t_v), 1]} \rceil - u_v \geq \sum_{v \in \Gamma} \frac{l_v}{x^*(t_v)} \text{ for all } v \in \Gamma. \tag{B.2}$$

Since the lower bound is less than or equal to the upper bound, let $u^* = \sum_{v \in \Gamma} u_v$, and combining equations (B.1) and (B.2) we get

$$\begin{aligned} \lceil \frac{f'(B(v))}{min[x^*(t_v), 1]} \rceil &\leq n, \\ \lceil \frac{f'(B(v))}{min[x^*(t_v), 1]} \rceil - u^* &\leq n - u^*, \\ \sum_{v \in \Gamma} \frac{l_v}{x^*(t_v)} &\leq n - u^*. \end{aligned}$$

Since if $2 - x^*(t_v) \geq f'(B(v))$, then $u_v = 0$, otherwise, $u_v = f'(B(v)) - l_v = f'(B(v)) - (2 - x^*(t_v)) \geq 0$, so $u_v \geq 0$ always holds. Then if $u_v \geq 1$, $max[u_v - 1, 0] = u_v - 1 \leq u_v$, if $0 \leq u_v \leq 1$, $max[u_v - 1, 0] = 0 \leq u_v$. So $\max[u_v - 1, 0] \leq u_v$ is always valid. So

$$\begin{aligned} \sum_{v \in \Gamma} max[f'(B(v)) - 1, 0] &= \sum_{v \in \Gamma} max[l_v - 1, 0] + \sum_{v \in \Gamma} max[u_v - 1, 0] \\ &\leq \sum_{v \in \Gamma} max[l_v - 1, 0] + \sum_{v \in \Gamma} u_v \\ &= \sum_{v \in \Gamma} max[l_v - 1, 0] + u^*. \end{aligned}$$

From the above analysis, we can form a knapsack problem as follows:

$$\begin{aligned} \text{Maximize} \quad & \textstyle\sum_{v\in\Gamma} max[l_v - 1, 0] \\ \text{Subject to:} \quad & \textstyle\sum_{v\in\Gamma} \frac{l_v}{x^*(t_v)} \leq n - u^* \end{aligned}$$

Solving this knapsack problem, we can get the upper bound of the objective function to be $(\sqrt{2}-1)^2(n-u^*)+u^*$. Since the fractional degree of a vertex v is replaced by a gadget with a set Γ_v of in-vertices that is at least $2+\sum_{w\in\Gamma_v} u_w$, so $u^* \leq 2(OPT_{LP}(G) - n)$. Hence,

$$\begin{aligned} \sum_{v\in\Gamma} max[l_v - 1, 0] + u^* &\leq (\sqrt{2}-1)^2(n - 2(OPT_{LP}(G) - n)) + 2(OPT_{LP}(G) - n) \\ &= (7 - 6\sqrt{2})n + 4(\sqrt{2}-1)OPT_{LP(G)}. \end{aligned}$$

- For (2): Consider the definition of f'', the flow pushed on all back arcs is $\sum_{w\in\Gamma_v}$max[1-$f'_{(v,w)}$,0] which equals the sum of (1-$f'_{(v,w)}$) where $f'_{(v,w)} < 1$, i.e. $\sum_{w\in\Gamma'_v}$(1-$f'_{(v,w)}$), where Γ'_v=$\{w \in \Gamma_v : f'_{(v,w)} < 1\}$. Let T_w be the set of nodes of G in the subtree of the undirected tree T rooted at the child of w$\in \Gamma'_v$, then by the definition of f',

$$f'_{(v,w)} = \sum_{a\in\delta(T_w)\setminus\delta(v)} min[x^*_a, 1] = x^*(\delta(T_w) \setminus \delta(v)).$$

Since all the edges $a \in \delta(T_w) \setminus \delta(v)$ have $x_a < 1$, so

$$\begin{aligned} \sum_{w\in\Gamma'_v} (1 - f'_{(v,w)}) &= \sum_{w\in\Gamma'_v} (1) - \sum_{w\in\Gamma'_v} x^*(\delta(T_w) \setminus \delta(v)) \\ &= |\Gamma'_v| - \sum_{w\in\Gamma'_v} x^*(\delta(T_w) \setminus \delta(v)). \end{aligned}$$

So we can get:

$$\begin{aligned} 2\sum_{w\in\Gamma'_v} x^*(\delta(T_w)\setminus\delta(v)) &= 2(\sum_{w\in\Gamma'_v}(1-f'_{(v,w)})-|\Gamma'_v|) \\ &= \sum_{w\in\Gamma'_v} x^*(\delta(T_w)) + x^*(\delta(\cup_{w\in\Gamma'_v} T_w \cup v)) - x^*(\delta(v)). \end{aligned}$$

Since for any cut in x^*, the value must be greater than or equal to 2 because of the subtour elimination constraints,

$$x^*(\delta(\cup_{w\in\Gamma'_v} T_w \cup v)) \geq \sum_{w\in\Gamma'_v} x^*(\delta(T_w)) + 2.$$

So

$$\begin{aligned} 2\sum_{w\in\Gamma'_v} x^*(\delta(T_w)\setminus\delta(v)) &\geq 2(\sum_{w\in\Gamma'_v} x^*(\delta(T_w))) + 2 - x^*(\delta(v)) \\ &\geq 2(|\Gamma'_v|+1) - x^*(\delta(v)). \end{aligned}$$

Thus

$$(|\Gamma'_v|+1) - x^*(\delta(v)) \leq \frac{x^*(\delta(v))}{2} - 1. \tag{B.3}$$

Applying this argument for each v we have

$$\begin{aligned} \sum_{v\in\Gamma} f''(B(v)) &= \sum_{v\in V}\sum_{w\in\Gamma'_v}(1-f'_{(v,w)}) \\ &\leq \sum_{v\in V}(\frac{x^*(\delta(v))}{2}-1) \\ &= OPT_{LP}(G) - n. \end{aligned}$$

Thus we get $\sum_{v\in\Gamma} f''(B(v)) \leq OPT_{LP}(G) - n$.

Summing the result in Claim 1 and Claim 2, we know the cost of f is at most $6(1-\sqrt{2})n + (4\sqrt{2}-3)OPT_{LP}(G)$.

Hence Theorem B.0.2 is proved.

□

Bibliography

[BEM07] Sylvia Boyd and Paul Elliott-Magwood. Operations for generating sep extreme points. Technical Report TR-2007-08, SITE, University of Ottawa, Ottawa, Canada, 2007.

[BIT13] Sylvia Boyd, Satoru Iwata, and Kenjiro Takazawa. Finding 2-factors closer to tsp tours in cubic graphs. *SIAM J. Discrete Math.*, 27(2):918–939, 2013.

[BM08] J.A. Bondy and U.S.R. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer, 1st edition, 2008.

[BP91] Sylvia C. Boyd and William R. Pulleyblank. Optimizing over the subtour polytope of the travelling salesman problem. *Math. Program.*, 49:163–187, 1991.

[BSvdSS11] Sylvia Boyd, René Sitters, Suzanne van der Ster, and Leen Stougie. The traveling salesman problem on cubic and subcubic graphs. In *Proceedings of the 15th International Conference on Integer Programming and Combinatoral Optimization*, IPCO'11, pages 65–77, Berlin, Heidelberg, 2011. Springer-Verlag.

[Chr76] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[CKK02] Béla Csaba, Marek Karpinski, and Piotr Krysta. Approximability of dense and sparse instances of minimum 2-connectivity, tsp and path problems. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '02, pages 74–83, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[CSS98] Joseph Cheriyan, András Sebö, and Zoltán Szigeti. An improved approximation algorithm for minimum size 2-edge connected spanning subgraphs. In Robert E. Bixby, E. Andrew Boyd, and Roger Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization*, volume 1412 of *Lecture Notes in Computer Science*, pages 126–136. Springer Berlin Heidelberg, 1998.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[Eve79] Shimon Even. *Graph Algorithms*. W. H. Freeman & Co., New York, NY, USA, 1979.

[Fle91] Herbert Fleischner. Chapter x: Algorithms for eulerian trails and cycle decompositions, maze search algorithms. In Herbert Fleischner, editor, *Eulerian Graphs and Related Topics*, volume 50 of *Annals of Discrete Mathematics*, pages X.1 – X.34. Elsevier, 1991.

[Gar85] R.S. Garfinkel. Motivation and modeling. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors, *The Traveling Salesman Problem*. John Wiley & Sons, 1985.

[GC85] Denis Naddef Gérard Cornuéjols, Jean Fonlupt. The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical Programming*, 33(1):1–27, 1985.

[GKP95] Michelangelo Grigni, Elias Koutsoupias, and Christos H. Papadimitriou. An approximation scheme for planar graph tsp. In *FOCS*, pages 640–645. IEEE Computer Society, 1995.

[GLS88] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.

[GLS05] David Gamarnik, Moshe Lewenstein, and Maxim Sviridenko. An improved upper bound for the tsp in cubic 3-edge-connected graphs. *Oper. Res. Lett.*, 33(5):467–474, September 2005.

[Goe07] Michel Goemans. Linear programming and polyhedral combinatorics. 2007.

[GSS11] Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 550–559, Washington, DC, USA, 2011. IEEE Computer Society.

[GT91] Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for general graph matching problems. *J. ACM*, 38(4):815–853, October 1991.

[Has31] Whitney Hassler. Non-separable and planar graphs. *Proceedings of the National Academy of Sciences of the United States of America*, 17(2):125–127, February 1931.

[HC73] Carl Hierholzer and Wiener Chr. Ueber die möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren. *Mathematische Annalen*, 6(1):30–32, 1873.

[HK71] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1(1):6–25, 1971.

[Hof60] A.J Hoffman. Some recent applications of the theory of linear inequalities to extremal combinatorial analysis. *Combinatorial Analysis*, 10:113 – 127, 1960.

[Huh04] Woonghee Tim Huh. Finding 2-edge connected spanning subgraphs. *Operations Research Letters*, 32(3):212 – 216, 2004.

[JRV03] Raja Jothi, Balaji Raghavachari, and Subramanian Varadarajan. A 5/4-approximation algorithm for minimum 2-edge-connectivity. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 725–734, 2003.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[Kar86] A.V. Karzanov. An algorithm for determining a maximum packing of odd-terminus cuts, and its applications. *Selected Topics in Discrete Mathematics Proceedings of the Moscow Discrete Mathematics Seminar 1972-1990*, 1986.

[KK01] Piotr Krysta and V.S. Anil Kumar. Approximation algorithms for minimum size 2-connectivity problems. In Afonso Ferreira and Horst Reichel, editors, *STACS 2001*, volume 2010 of *Lecture Notes in Computer Science*, pages 431–442. Springer Berlin Heidelberg, 2001.

[KV94] Samir Khuller and Uzi Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM*, 41(2):214–235, March 1994.

[Lov85] László Lovász. Computing ears and branchings in parallel. volume 0, pages 464–467, Los Alamitos, CA, USA, 1985. IEEE Computer Society.

[MR86] Gary L. Miller and Vijaya Ramachandran. Efficient parallel ear decomposition with applications. unpublished manuscript, MSRI, Berkeley, CA., January 1986.

[MS11] Tobias Mömke and Ola Svensson. Approximating graphic tsp by matchings. *CoRR*, abs/1104.3090, 2011.

[Muc11] Marcin Mucha. Improved analysis for graphic tsp approximation via matchings. *CoRR*, abs/1108.1130, 2011.

[R11] Ravi. R. A 4/3-approximation algorithm for tsp on k-regular k-edge-connected graphs for k=4,5. 2011.

[Sch95] Alexander Schrijver. Handbook of combinatorics (vol. 2). chapter Polyhedral Combinatorics, pages 1649–1704. MIT Press, Cambridge, MA, USA, 1995.

[Sch03] Alexander Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Algorithms and Combinatorics. Springer, 2003.

[SLKL85] D.B. Shmoys, J.K. Lenstra, A.H.G.R. Kan, and E.L. Lawler. *The Traveling Salesman Problem*. Wiley Interscience Series in Discrete Mathematics. John Wiley & Sons, 1985.

[SV12] András Sebő and Jens Vygen. Shorter tours by nicer ears: 7/5-approximation for graphic tsp, 3/2 for the path version, and 4/3 for

two-edge-connected subgraphs. Technical Report 121042, Research Institute for Discrete Mathematics, University of Bonn, Cahiers Leibniz no. 198, Laboratoire G-SCOP, Grenoble., 2012. Accepted for publication in Combinatorica.

[SW90] D. B. Shmoys and D. P. Williamson. Analyzing the held-karp tsp bound: A monotonicity property with application. *Inf. Process. Lett.*, 35(6):281–285, September 1990.

[SWvZ12] Frans Schalekamp, David P. Williamson, and Anke van Zuylen. A proof of the boyd-carr conjecture. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1477–1486. SIAM, 2012.

[Tar85] Éva Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, July 1985.

[TH01] Ronald L.Rivest Clifford Stein Thomas H.Cormen, Charles E.Leiserson. *Introduction to Algorithms*. The MIT Press, 2 edition, 2001.

[VV00] Santosh Vempala and Adrian Vetta. Factor 4/3 approximations for minimum 2-connected subgraphs. In *Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization*, APPROX '00, pages 262–273, London, UK, UK, 2000. Springer-Verlag.

Index