

UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Raul Wagner Martins Costa

An algorithmic study of the Steiner Multicycle Problem

Belo Horizonte
2024

Raul Wagner Martins Costa

An algorithmic study of the Steiner Multicycle Problem

Final Version

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Prof. Dr. Phablo Fernando Soares Moura

Belo Horizonte
2024

Costa, Raul Wagner Martins.

C837a An algorithmic study of the Steiner Multicycle Problem
[recurso eletrônico] / Raul Wagner Martins Costa. – 2024.

1 recurso online (87 f. il., color.) : pdf.

Orientador: Phablo Fernando Soares Moura.

Dissertação (mestrado) - Universidade Federal de Minas
Gerais, Instituto de Ciências Exatas, Departamento de Ciência
da Computação.

Referências: f. 82 - 87.

1. Computação – Teses. 2. Algoritmos de aproximação –
Teses. 3. Teoria dos grafos – Teses. 4. Modelagem de dados –
Transporte de cargas (Computação) – teses. I. Moura, Phablo
Fernando Soares. II. Universidade Federal de Minas Gerais,
Instituto de Ciências Exatas, Departamento de Ciência da
Computação. III. Título.

CDU 519.6*62(043)



UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

An Algorithmic study of the Steiner Multicycle Problem

RAUL WAGNER MARTINS COSTA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores(a):

Documento assinado digitalmente



PHABLO FERNANDO SOARES MOURA
Data: 14/09/2024 15:01:37-0300
Verifique em <https://validar.iti.gov.br>

Prof. PHABLO FERNANDO SOARES MOURA - Orientador
Departamento de Ciência da Computação - UFMG

Documento assinado digitalmente



MARCIO COSTA SANTOS
Data: 16/09/2024 13:51:27-0300
Verifique em <https://validar.iti.gov.br>

PROF. MÁRCIO COSTA SANTOS
Departamento de Ciência da Computação - UFMG

Documento assinado digitalmente



CARLA NEGRI LINTZMAYER
Data: 15/09/2024 11:52:43-0300
Verifique em <https://validar.iti.gov.br>

PROFA. CARLA NEGRI LINTZMAYER
Centro de Matemática, Computação e Cognição - Universidade Federal do ABC

Belo Horizonte, 21 de fevereiro de 2024.

Este trabalho é dedicado a Aggeu Vieira Marques Filho.

Acknowledgments

Gostaria de expressar minha profunda gratidão aos meus pais Wagner e Michelle e ao meu irmão Michel pelo companherismo e suporte ao longo desta jornada. Não poderia deixar de agradecer ao meu orientador, Phablo Moura, cuja paciência, orientação e incentivo foram essenciais ao longo deste processo. Um agradecimento especial ao meu mentor e amigo, Pedro Pazzini, cujo apoio foi fundamental para o meu início nesta jornada que é o mestrado. Por fim, sou imensamente grato à minha companheira de vida Maria Vitória pela paciência, suporte e incentivo constante ao longo de todo o processo.

“We live only to discover beauty. All else is a form of waiting.”
(Khalil Gibran)

Resumo

No Problema de Multiciclo de Steiner (SMCP), temos um grafo G , uma função de custo $c: E(G) \rightarrow \mathbb{R}_{\geq}$ e uma coleção de conjuntos de terminais $\{T_1, \dots, T_k\}$, onde para cada a em $[k]$ T_a é um subconjunto de $V(G)$ e esses conjuntos de terminais são disjuntos. O problema consiste em encontrar um conjunto de custo mínimo de ciclos disjuntos em vértices tal que, para cada a em $[k]$, todos os vértices de T_a pertençam a um mesmo ciclo. Existem aplicações relacionadas a problemas de roteamento que podem ser traduzidas em instâncias do SMCP. Mais precisamente, o SMCP modela um cenário de transporte de cargas pequenas, onde diversas empresas devem visitar periodicamente os locais de coleta e entrega. Para reduzir seus custos individuais, essas empresas podem colaborar para criar rotas compartilhadas que visitem vários locais de coleta e entrega, de forma a reduzir os custos totais de transporte. Claramente, o SMCP é uma generalização do Problema do Caixeiro Viajante e, portanto, não admite um esquema de aproximação em tempo polinomial (PTAS), mesmo no caso métrico. Um ponto positivo é que a literatura para o SMCP descreve um PTAS para instâncias euclidianas. Nesse trabalho, generalizamos este resultado mostrando um PTAS para SMCP em grafos de largura de árvore limitada e, a partir deste, derivamos um PTAS para grafos de *genus* limitado. O algoritmo proposto é essencialmente uma adaptação do PTAS para o Problema da Floresta de Steiner em grafos de *genus* limitado. Além disso, implementamos o algoritmo 3-aproximado para SMCP proposto por [Fernandes et al. \[2024\]](#) em grafos métricos completos e executamos experimentos computacionais. Os custos das soluções produzidas por este algoritmo são em média 34% piores do que as soluções ótimas correspondentes, o que é consideravelmente melhor do que a garantia teórica. Por outro lado, o tempo de execução da implementação é em geral pior do que as outras heurísticas apresentadas na literatura.

Palavras-chave: algoritmos de aproximação; teoria dos grafos; genus limitado; Multiciclos de Steiner; experimentos computacionais.

Abstract

In the Steiner Multicycle Problem (SMCP), we have a graph G , a cost function $c: E(G) \rightarrow \mathbb{R}_{\geq}$ and a collection of terminal sets $\{T_1, \dots, T_k\}$, where for each a in $[k]$ T_a is a subset of $V(G)$ and these terminal sets are pairwise disjoint. The problem consists of finding a minimum cost set of vertex-disjoint cycles such that, for each a in $[k]$, all vertices of T_a belong to the same cycle. There are applications related to routing problems that can be translated into SMCP instances. More precisely, SMCP models a less-than-truckload scenario, where multiple companies must periodically visit pickup and delivery locations. To reduce their individual costs, these companies can collaborate to create shared routes that visit multiple pickup and delivery locations to reduce total transportation costs. Clearly, SMCP is a generalization of the Traveling Salesman Problem and thus does not admit a polynomial-time approximation scheme (PTAS), even in the metric case. On the positive side, the literature for this problem describes a PTAS for Euclidean instances. In this work, we generalize this result by showing a PTAS for SMCP in graphs of bounded treewidth and, from this, we derive a PTAS for graphs of bounded *genus*. The proposed algorithm is essentially an adaptation of the PTAS for the Steiner Forest Problem on graphs of bounded *genus*. Furthermore, we implemented the 3-approximate algorithm for SMCP proposed by [Fernandes et al. \[2024\]](#) on complete metric graphs and performed computational experiments. The solution costs produced by this algorithm are on average 34% worse than the corresponding optimal solutions, which is considerably better than the theoretical guarantee. On the other hand, the running time of the implementation is generally worse than other heuristics presented in the literature.

Keywords: approximation algorithms; graph theory; bounded genus; Steiner Multicycle; computational experiments.

List of Figures

1.1	An example of a feasible solution for the SMCP considering the set of pairs of terminals $\{\{t_1, t'_1\}, \{t_2, t'_2\}, \{t_3, t'_3\}\}$. The solution is represented by the edges in bold and corresponds to the multiset $\{e_1, e_2, e_3, e_4, e_5, e_6, e_9, e_{10}, e_{11}, e_{10}, e_{11}\}$. The weights of the edges were omitted for simplicity.	16
2.1	Example of a graph and its tree decomposition. (Song and Yu [2015]).	21
4.1	Instances randomly generated with 16 agents, in (a) a 1 x 1 frame with no wall, in (b) a 3 x 3 frame with 0% wall, and in (c) a 3 x 3 frame with 30% wall.	36
4.2	Number of instances per optimality gap. An optimality gap of 1.6 means that the solution cost is 60% more than the cost of the relaxation.	38
5.1	View of solution M on bag B_i . This solution induces a partition $\pi_i(M) = \{\{A_{i1}, A_{i2}\}, \{A_{i3}\}\}$ of A_i . If this partition belongs to Π_i , then M conforms to Π_i	43
5.2	Example of connection between K_{j^*} and K_j . Note that $K_{j^*} < K_j$. The thick red line represents the connection created between both components. This connection consists of the shortest path between both components (it does not necessarily go through edges in B_i).	45
6.1	In the top figure, the graph G is depicted with the approximated solution M and an optimal solution M^* represented with red and green lines, respectively. The bottom figure illustrates G contracted by M , where each vertex $\{c_1, c_2, c_3\}$ corresponds to a component of M . The graph L consists of the red line and the vertices $\{c_1, c_2, c_3\}$	63
6.2	Example of cutting process using a subgraph T to form a face H inside the graph. (Borradaile et al. [2014]).	65
6.3	Example of a cut in a graph with <i>genus</i> greater than zero. (Borradaile et al. [2014]).	66
6.4	Example of $loop(T, e)$. The tree T is rooted at the vertex F and is represented by the black (non-opaque) edges. The edge e is represented in green and the paths between e and the vertex F are displayed in red. The $loop(T, e)$ is composed of the union of the red and green edges.	66
6.5	Construction of strips in Mortar Graph (Borradaile et al. [2009]).	69
6.6	Construction of a brick. (Borradaile et al. [2009]).	70

6.7	Mortar Graph. (Borradaile et al. [2009]).	72
6.8	Cleavings illustrated (Borradaile and Klein [2016]).	74

List of Tables

4.1	Summary of results by class.	37
4.2	Participation of Gomory-Hu trees calculation on total execution time	38

List of Algorithms

1	SMCP 3-approximation	35
2	PC-Clustering	58
3	PC-Partition	62
4	SMCP-PTAS	77

Contents

1	Introduction	15
2	Definitions	18
2.1	Graph Notations and Concepts	18
2.1.1	Cycles in the SMCP	21
2.2	Graph Planarity and Genus	22
2.3	Classes of Optimization Problems	23
3	Related Work	26
3.1	Travelling Salesman Problem	26
3.2	Steiner-related problems	28
3.2.1	Steiner Tree Problem	29
3.2.2	Steiner Cycle Problem	30
3.2.3	Multiple Steiner Traveling Salesman Problem	31
3.2.4	Steiner Forest Problem	31
3.2.5	Steiner Multicycle Problem	32
4	Computational Experiments	34
4.1	3-approximation Algorithm	35
4.2	Instances	36
4.3	Computational Results	37
5	PTAS for Graphs of Bounded Treewidth	40
5.1	Groups	40
5.2	Notation and definitions	41
5.3	Constructing the Partitions	42
5.4	Conforming Solutions	46
6	PTAS for Graphs of Bounded Genus	54
6.1	Prize Collecting Partition	54
6.1.1	Prize-Collecting Clustering	55
6.1.2	Proof of PC-Partition Theorem	62
6.2	Spanner	64
6.2.1	Mortar Graph	65
6.2.1.1	Cut graph construction	65

6.2.1.2	Strips	68
6.2.1.3	Bricks	70
6.2.1.4	Portals	71
6.2.2	Spanner construction	72
6.3	PTAS for Graphs of Bounded Genus	76
7	Conclusion	79
	References	80

Chapter 1

Introduction

The Steiner Multicycle Problem (SMCP) was introduced by [Pereira et al. \[2018\]](#) in the context of vehicle routing where multiple companies must visit pickup and delivery locations. That way, a group of companies can collaborate to realize the necessary deliveries and reduce the total cost of transportation. Notably, one assumes that the companies must make the deliveries periodically, and the size of each delivery is much smaller than the vehicles' capacities. The interested reader may find more details about this application in [Pereira et al. \[2018\]](#).

The problem SMCP has as input an undirected graph G , a function $c: E(G) \rightarrow \mathbb{R}_{\geq}$ that associates a cost to each edge of G , and a set $\{T_1, \dots, T_k\}$ of pairwise disjoint subsets of vertices of G (a.k.a. **terminal sets**). The goal of the problem is to find a minimum-cost set of cycles $\mathcal{C}$, such that each terminal set T_i is contained within the same cycle.

A solution to the Steiner Multicycle Problem can be represented by a single multiset that contains all the edges forming the cycles in the solution. Moreover, since a multiset of edges and the cycles induced by these edges are equivalent, throughout this work, we refer to the solution as both the multiset of edges and the subgraph of G induced by those edges interchangeably.

The solution cost is defined as the sum of all associated costs of the edges in the solution multiset.

The problem is defined next. An instance of SMCP and a feasible solution are depicted in Figure 1.1.

STEINER MULTICYCLE PROBLEM (SMCP)

Instance: A graph G , $c: E(G) \rightarrow \mathbb{R}_{\geq}$, and a set $\{T_1, \dots, T_k\}$ of pairwise disjoint subset of vertices of G (i.e. the terminals).

Goal: Find in G a minimum cost set of cycles such that all terminals in T_i belong to the same cycle, for each $i \in [k]$.

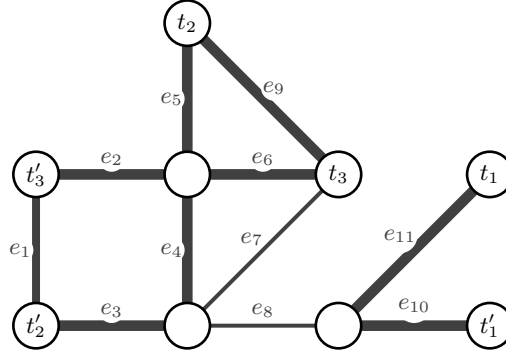


Figure 1.1: An example of a feasible solution for the SMCP considering the set of pairs of terminals $\{\{t_1, t'_1\}, \{t_2, t'_2\}, \{t_3, t'_3\}\}$. The solution is represented by the edges in bold and corresponds to the multiset $\{e_1, e_2, e_3, e_4, e_5, e_6, e_9, e_{10}, e_{11}, e_{10}, e_{11}\}$. The weights of the edges were omitted for simplicity.

Pereira et al. [2018] studied the problem for complete metric graphs (i.e., complete graphs that guarantee the triangular inequality for the edges costs). The authors presented a 4-approximation algorithm, a heuristic algorithm, and a mixed-integer linear formulation for the problem. Moreover, the authors performed computational experiments comparing the proposed heuristic, called *Refinement Search*, to a GRASP implementation. In summary, the results showed that the proposed heuristic reached better quality results in less time.

Pereira et al. also introduced a restricted version of the Steiner Multicycle Problem (R-SMCP) in which every terminal set has only two vertices, one for pickup and the other for delivery. Lintzmayer et al. [2020] showed an equivalence between the two problems – SMCP and R-SMCP – by proposing a simple transformation from instances of the SMCP to R-SMCP, which we present below.

Let $\mathcal{T} = \{t_1, \dots, t_\ell\}$ be a set of terminals, and create ℓ new vertices $\{u_1, \dots, u_\ell\}$. For each $i \in \{1, \dots, \ell\}$, we create an edge $\{t_i, u_i\}$ of cost zero. We define $\{\{u_1, t_2\}, \dots, \{u_{\ell-1}, t_\ell\}, \{u_\ell, t_1\}\}$ as terminals pairs of an instance of R-SMCP. Note that the set of edges that forms a solution of SMCP with terminals of $\mathcal{T}$ have the same cost of a solution of R-SMCP after the addition of the new vertices. Given that equivalence, throughout this work, we assume that each terminal set in the SMCP contains exactly two vertices; i.e., we will refer to the R-SMCP as SMCP and denote terminal sets as terminal pairs.

Lintzmayer et al. [2020] proposed the first PTAS for the SMCP. The algorithm is a randomized PTAS restricted to the Euclidean plane. In this context, the Euclidean SMCP encompasses a set of terminal pairs distributed across a plane and aim to calculate the line segments that connect the points with the least cost, considering that the same line segment might be crossed more than once in the cost function.

Fernandes et al. [2024] proposed a 3-approximation for complete metric graphs, which is based on a 2-approximation for the Survivable Network Design Problem and the concept of T-joins, a generalization of the well-known perfect matching. This algorithm

is inspired by the results obtained for the metric Travelling Salesman Problem (TSP) by Christofides [1976].

It is also essential to notice that, even for the restricted case, the Steiner Multicycle Problem is $\mathcal{NP}$ -hard. This can be verified by reducing the TSP to the R-SMCP using a strategy similar to the one presented by Lintzmayer et al.

In this work, we propose a **polynomial-time approximation scheme** (PTAS) for the SMCP on *bounded treewidth graphs*. From this result, we also present a PTAS on *bounded genus graphs*. This class of graphs is a generalization of the well-known class of planar graphs.

This work is structured as follows. Chapter 2 introduces some basic definitions in Graph Theory and Approximation Algorithms, as well as some definitions used to present and prove the results of this work. Chapter 3 overviews the literature for SMCP and related problems. Chapter 4 presents some experimental results of the 3-approximation algorithm proposed by Fernandes et al. [2024]. Chapter 5 proves a PTAS for bounded treewidth graphs, which is an essential result to prove the PTAS for bounded genus graphs. After that, Chapter 6 presents some tooling necessary for achieving the PTAS for bounded genus graphs and proves the PTAS for this class of graphs. Finally, Chapter 7 delivers general remarks on this work and proposes possible paths for future work.

Chapter 2

Definitions

This chapter presents the definitions and notations used throughout this work. The definitions shown are based on [Bondy and Murty \[2008\]](#) and [Diestel \[2005\]](#).

2.1 Graph Notations and Concepts

A graph is a pair $G = (V, E)$ of sets satisfying $E \subseteq [V]^2$; that is, the elements of E are subsets of V with two elements. The elements of V are called vertices and of E are called edges.

The set of vertices of a graph G are also denoted as $V(G)$ and the set of edges as $E(G)$. We express membership of a vertex v to a graph G with the notation $v \in V(G)$, and similarly, for an edge e , we denote $e \in E(G)$. Alternatively, we can use $v \in G$ or $e \in G$ when the context makes it clear that we are referring to vertices or edges, respectively.

A vertex v is **incident** to an edge e if $v \in e$; so e is an edge connected to v . The two vertices incidents to e are called the **endpoints** of e . Another notation commonly used to represent an edge $\{v, w\}$ is by vw simply, where $v, w \in V(G)$ are the endpoints.

The number of vertices of a graph is referred to as the graph's **order**, denoted by $|G|$. The number of edges is referred to as its **size** and is denoted by $||G||$.

Two vertices u, v of G are **adjacent** or **neighbors** if there is $e \in E(G)$ such that u and v are its endpoints. Equivalently, two edges e and f are adjacent if they have one of their endpoints in common.

Let $v \in V(G)$ be a vertex. We denote as $N(v)$ the **open neighborhood** of v , that is, the set of all vertices $w \in V(G)$ such that w is adjacent to v . The **closed neighborhood** of v , denoted as $N[v]$ is $N(v) \cup \{v\}$.

If there is an edge $uv \in E(G)$ for any pair of vertices $u, v \in V(G)$, then we say that G is a **complete** graph. A complete graph of n vertices is denoted as K^n .

The **degree** of a vertex v is the number of edges of G incident to v , denoted by $d(v)$. If v is a vertex such that $d(v) = 0$, we say that v is an **isolated vertex**.

The number $\delta(G) := \min\{d(v) : v \in V(G)\}$ is the **minimum degree** of G ; the number $\Delta(G) := \max\{d(v) : v \in V(G)\}$ is the **maximum degree** of G .

We say that a graph H is a **subgraph** of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$, and denote this by $H \subseteq G$. If for any pair of vertices $v, w \in H \subset G$, the statement $vw \in E(H)$ is valid if, and only if, $vw \in E(G)$, then we say that H is an **induced subgraph** of G , denoted as $G[H]$. Moreover, we say that H is a **spanning subgraph** of G if $V(H) = V(G)$.

We define $c : E(G) \rightarrow \mathbb{R}_{\geq}$, for $e \in E(G)$ as the **cost** of e . Naturally, $c(H)$, for $H \subseteq G$, is the sum of the cost of the edges of the subgraph H . Let $C = \{(v_1, v_2, \dots, v_k)\}$ be a cycle of G . We define $c(C) = \sum_{i=1}^{k-1} c(v_i v_{i+1})$.

A **path** in a graph G is a sequence of distinct vertices $P = (v_1, v_2, \dots, v_k)$, $P \subseteq V(G)$, such that, for every pair of consecutive vertices of P , there is an edge in $E(G)$ that connects these vertices. That is, for every v_i, v_{i+1} in P , there is $v_i v_{i+1} \in E(G)$.

Let $C = (v_1, v_2, \dots, v_k)$ be a sequence of adjacent vertices such that $|C| \geq 3$ and the endpoints of the sequence C are equal, that is, $v_1 = v_k$, then we say that C is a **cycle**. The **cost** of a cycle equals the sum of the costs of its edges.

A **walk** (of length k) is a non-empty alternating sequence $(v_0, e_0, v_1, e_1, \dots, e_{k-1}, v_k)$ of vertices and edges such that $e_i = \{v_i, v_{i+1}\}$ for all $i < k$. In particular, in the case $v_0 = v_k$, we call it a **closed walk**. Note that, in a walk, vertices and edges can be visited more than once.

Given $u, v \in V(G)$, if there is a path between u and v in G , we say that the vertices are **connected**.

If there is a partition of $V(G)$ into subsets $V_1, V_2, \dots, V_w$ such that two vertices u and v are connected if, and only if, u and v belong to the same subset V_i , then we say that the subgraphs $G[V_1], G[V_2], \dots, G[V_w]$ are the **components** of G . If G contains a single component, we say that G is **connected**, otherwise G is **disconnected**.

Let $e \in E(G)$. We define as **contracting** the operation on e which consists on replacing both endpoints of e with a new vertex v_e and connecting v_e to e endpoint's neighbors.

We denote the graph obtained from G by contracting the edge $e \in E(G)$ at a new vertex v_e , which becomes adjacent to all old neighbors of the endpoint vertices of e , as G/e .

Given a graph G and a subgraph H of G , we **contract** H into a single vertex v , generating a new graph $G^* := G/H$. All vertices of $G - H$ adjacent to at least two vertices from H are connected to v in G^* with multiple edges. We also define the process of **uncontracting** v as replacing it with the original subgraph H and reconstructing the previous connections with the neighboring vertices of H in G .

Given a graph G and a graph H , we say that H is a **minor** of G if, and only if, H can be obtained by a series of edges contractions on G .

A collection $\mathcal{S}$ is said to be **laminar** if and only if for any two sets $C_1, C_2 \in \mathcal{S}$ we have $C_1 \subseteq C_2$, or $C_2 \subseteq C_1$, or $C_1 \cap C_2 = \emptyset$. Suppose $\mathcal{C}$ is a partition of a ground set V . Then, $\mathcal{C}(v)$ denotes, for each $v \in V$, the set $C \in \mathcal{C}$ that contains v .

For vertices $v, w \in V(G)$ we define the **distance** between vertices v and w , denoted by $\text{dist}_G(v, w)$, as the cost of a shortest path between v and w that only uses edges of G . If such a path does not exist, by convention, we consider $\text{dist}_G(v, w) = +\infty$. We generalize the notation for distances between vertices and edges. Let $v \in V(G)$ and $uw \in E(G)$. Define the distance between the vertex v and the edge uw as $\text{dist}_G(v, uw) := \min\{\text{dist}_G(v, u), \text{dist}_G(v, w)\} + c(uw)$.

A graph F without cycles is a **forest**. A connected forest is a **tree**. Let T be a tree and $v \in T$ a vertex. If $d(v) = 1$, we say that v is a **leaf** of T , all other vertices are called inner vertices of T . A tree is rooted if one of its vertices has been chosen as **root**. Naturally, each forest component is a tree.

Let T be a rooted tree with root $r \in V(T)$, let $v \in V(T)$. Note that there is only a single path between r and v in T . Let P be this unique path between v and r . We say that $w \in V(T)$ is the **parent vertex** of v if $w \in P$ and $w \in N(v)$. Equivalently, we say that v is a **child vertex** of w . Note that a vertex can have multiple children but only one parent.

Given a connected graph G , we say that a **shortest-path tree** SPT rooted at a vertex v of $V(G)$ is a spanning tree of G such that the path distance from root v to any other vertex $u \in V(SPT)$ is a shortest path from v to u in G , i.e., $\text{dist}_{SPT}(u, v) = \text{dist}_G(u, v)$.

As defined by [Robertson and Seymour \[1986\]](#), a **tree decomposition** of G is a pair (T, B) in which T is a tree and $B = \{B_i : i \in V(T)\}$ is a family of subsets of $V(G)$, so that:

- $\bigcup_{i \in V(T)} B_i = V(G)$;
- For each edge $uv \in E(G)$, there is a $i \in V(T)$ such that $u, v \in B_i$;
- For each $v \in V(G)$, the set of vertices $\{i \in V(T) : v \in B_i\}$ forms a subtree of T .

To differentiate from the original graph G , we call the vertices of T **nodes**, where each node i has a corresponding **bag** of vertices B_i . An example can be seen in Figure 2.1.

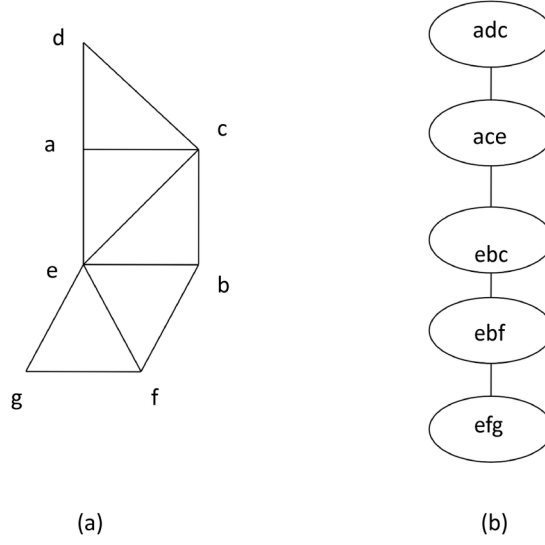


Figure 2.1: Example of a graph and its tree decomposition. (Song and Yu [2015]).

The **width** of a tree decomposition is the size of its largest bag minus one. The **treewidth** of a graph G , denoted with $tw(G)$, is the smallest width of a tree decomposition of G .

To facilitate application in algorithms, we focus on a more restricted class of decompositions. We say that a tree decomposition (T, B) is **nice** when T is a rooted tree and each $i \in V(T)$ belongs to one of the following classes:

- i is a *leaf node* if it has no children;
- i is a *union node* if i has exactly two children i_1, i_2 and it holds $B_i = B_{i_1} = B_{i_2}$;
- i is an *introduction node* if it has a single child i' and it holds $B_i = B_{i'} \cup \{v\}$ for some vertex $v \in V(G)$;
- i is a *forgetting node* if it has a single child i' and it holds $B_i = B_{i'} \setminus \{v\}$ for some vertex $v \in V(G)$.

Given two bags B_i and B_j , we say that B_j is a **descendant** of B_i if it is separated from the root bag by B_i .

2.1.1 Cycles in the SMCP

In this work, we overload the term “cycle” to denote a closed walk in a graph or, equivalently, the multiset of edges that form the walk. A noteworthy observation is that

all vertices along the walk have even degrees, taking into consideration the repetitions of edges. Conversely, we call “simple” a cycle without edges repetition.

2.2 Graph Planarity and Genus

Informally, we characterize as **planar** a graph G that can be drawn on a plane without its edges intersecting outside their endpoints. Thus, the graph can be seen as points on a surface (vertices) with arcs (edges) connecting the points in such a way that arcs do not intersect outside a point. The surface region surrounded by arcs is called a **face** of the graph. Moreover, the outer region of the graph is the **outer face**, while all other faces are **inner faces** of G . The **boundary** of G is the set of edges that separate the outer face of G from the rest of the graph, is denoted by $\partial(G)$.

Formally, [Diestel \[2005\]](#) defines a planar graph as a pair (V, E) of finite sets with the following properties:

- $V \subseteq \mathbb{R}^2$;
- every edge is an arc between two vertices;
- different edges have a different set of extreme points;
- The interior of an edge does not contain vertices or intersections with other edges.

For any planar graph, the set $\mathbb{R}^2 \setminus G$ is open, and its regions are the faces of G .

The intuitive idea of “drawing” a planar graph into a “plane” is what we call a **planar embedding** of G . According to [Diestel \[2005\]](#), an embedding in the plane of an (abstract) graph G is an isomorphism between G and a **plane graph** $\tilde{G}$. The latter is referred to as **drawing** of G .

[Bondy and Murty \[2008\]](#) showed that a planar embedding of a graph G exists if and only if G is also embeddable on the sphere. This idea will be useful to generalize the concept of embedding graphs to other surfaces.

Formally, they define a **surface** as a 2-dimensional manifold. We will skip a more detailed presentation of the topological definitions and properties of this structure, as for the reader it suffices to know that a sphere is a surface, as well as a torus and other constructions formed by pushing additional “holes” in the sphere. The number of “holes” in the sphere is what we call the *genus* of the surface.

Informally, we say that the **genus** of a graph is the smallest number g such that the graph can be drawn on a sphere with g holes without edges crossing. From this idea, a planar graph has genus 0.

Interestingly, the problem of determining the genus of a graph is known to be $\mathcal{NP}$ -complete (Thomassen [1989]). However, as presented by Kawarabayashi et al. [2008], the problem becomes treatable when the genus g is fixed; that is, we can determine in polynomial time whether a graph can be embedded in a surface with genus g , considering g constant. As a secondary result, the authors also present a linear time algorithm, which computes the genus and constructs minimum genus embeddings of graphs of bounded treewidth.

For a given planar graph G , we say that H is its **dual** if there is a bijective function that maps the faces of G to the vertices of H and, for each edge $e \in E(G)$ that separates the faces f and f' of G , we have an edge that connects the vertices associated to f and f' in H .

The concept of **faces** can be generalized to higher *genus* graphs when we embed the graph on a surface of same genus.

2.3 Classes of Optimization Problems

In combinatorial optimization problems, we want to obtain the best solution from a finite, but potentially large, set of possible solutions. Ferreira et al. [2001] defined an optimization problem with the following elements: a set of instances, a set $\text{Sol}(I)$ of viable solutions for a given instance I , and a function $\text{val}(I, S)$ that associates to each instance and solution $S \in \text{Sol}(I)$ a non-negative rational value. The solution to the optimization problem is the S that minimizes/maximizes $\text{val}(I, S)$.

More formally, for a given instance I there is a solution $S^* \in \text{Sol}(I)$ such that $\text{val}(I, S^*) \leq \text{val}(I, S)$ for all $S \in \text{Sol}(I)$, considering a minimization scenario. The idea follows equivalently for a maximization problem. We denote $\text{OPT} := \text{val}(I, S^*)$ as the value of the optimal solution of I . In particular, for the problems discussed in this work, we denote as $\text{OPT}_{\mathcal{T}}(G)$ the value of an optimal solution in the graph G considering a set of pairs of terminals $\mathcal{T}$.

There is an intimate relationship between the classes of optimization problems and the well-known classes of decision problems $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$ -hard, $\mathcal{NP}$ -complete. There are optimization problems for which there are limitations of the approximability of their solutions by polynomial algorithms, considering $\mathcal{P} \neq \mathcal{NP}$.

That said, optimization problems are classified according to their degree of ap-

proximability. By approximability, we observe a scenario in which we give up finding an optimal solution in favor of finding a solution that, while sub-optimal, can be found efficiently and maintain quality guarantees. We will consider and detail the classes PO, PTAS (and its variants QPTAS, FPTAS, and EPTAS), APX, and NPO, as described by Ferreira et al. [2001].

According to Ferreira et al., the NPO class, which is an extension of the $\mathcal{NP}$ class for optimization problems, is composed of the problems where:

- There is a polynomial function p such that the size of the solution S is less or equal to the size of the instance I applied on p , for every instance I of the problem and every feasible solution S of I ;
- There is a polynomial time algorithm that decides whether a given word is a valid representation of an instance of the problem;
- There is a polynomial time algorithm that decides whether a given object is a feasible solution for a given instance of the problem;
- There is a polynomial time algorithm that calculates $\text{val}(I, S)$, given I and S .

We denote as PO the class of problems treatable in polynomial time. This class is an extension of the $\mathcal{P}$ class for optimization problems. Therefore, given a problem π , we say that $\pi \in PO$ if there is a polynomial time algorithm that calculates an optimal solution for each instance I of π . Examples of problems in this class are the Shortest Path and Minimum Spanning Tree problems.

Consider an optimization problem. Let A be an algorithm that for every instance I of the problem returns a feasible solution $A(I)$ of I . If the problem is one of minimization and $\text{val}(I, A(I)) \leq \alpha \text{OPT}$ holds for every instance I , then we say that A is a α -**approximation** for the problem. We define α as the **approximation ratio** of the algorithm.

The APX class comprises optimization problems in NPO for which there is an α -approximation for some constant α . One of the most famous algorithms that provide this type of approximation is the Christofides Algorithm, developed by Christofides [1976] to the TSP problem restricted to metric graphs and which guarantees an approximation ratio of $\alpha = 1.5$.

For a given optimization problem, we define as an **approximation scheme** an algorithm that receives an instance I of the problem as well as a parameter $0 < \epsilon < 1$ and returns a solution S such that $\text{val}(I, S) \leq (1 + \epsilon) \text{OPT}$, for a minimization problem. We say that an algorithm is a **polynomial-time approximation scheme** (PTAS) if it has polynomial time for every fixed ϵ . Furthermore, an algorithm is a **fully polynomial-time**

approximation scheme, or FPTAS, if its time is also polynomial in $1/\epsilon$. In contrast, a PTAS algorithm which is not FPTAS is exponential in $1/\epsilon$.

We will briefly comment on two other PTAS classes of interest. An **efficient polynomial-time approximation scheme** (EPTAS) is a more relaxed version of the FPTAS, where the complexity on $1/\epsilon$ does not need to be polynomial. I.e., a PTAS algorithm with time complexity $\mathcal{O}(c^{1/\epsilon} n^k)$, for c and k constants, is an EPTAS.

Another class worth mentioning is the **quasi-polynomial-time approximation scheme** (QPTAS), which allows a sub-linear factor as an exponent, i.e., an algorithm with time-complexity $\mathcal{O}(c^{\log n})$ for every fixed ϵ and where c is a constant.

From the definitions, it follows that

$$PO \subseteq FPTAS \subseteq PTAS \subseteq APX \subseteq NPO.$$

Chapter 3

Related Work

The Steiner Multicycle Problem (SMCP) is related to vehicle routing problems, particularly pickup and delivery problems. The literature for this class of problems is vast and considers multiple different restrictions and scenarios (Parragh et al. [2008]).

The SMCP stands between (and generalizes) two big classes of problems that are of much interest to the optimization and theoretical computing communities: the Traveling Salesman Problem (TSP) and the Steiner Tree Problem (STP).

We start this chapter by diving into the literature on the TSP, and some of its variants. We then move to the STP and discuss some of its variants that are more related to this study. Our literature review of these problems focuses on approximation algorithms, and more specifically on PTASes.

3.1 Travelling Salesman Problem

One of the most studied vehicle routing optimization problem is the Travelling Salesman Problem or TSP. In this problem, we aim to find a minimal-cost Hamiltonian cycle (i.e., a closed cycle that visits each vertex exactly once) in a given graph.

As mentioned in Chapter 1, we can draw a link between the TSP and the SMCP, since an instance of the TSP can be converted in polynomial time to an instance of the SMCP. That is, the TSP is a particular case of the SMCP.

The **Christofides Algorithm**, proposed by Christofides [1976], is one of the most famous approximation algorithms for the TSP. This algorithm is a $3/2$ -approximation on metric graphs.

Williamson and Shmoys [2011] present the following summary of the algorithm. Given a metric graph G , we calculate its minimum spanning tree M . Let O be the set of odd-degree vertices in M . By the Handshaking Lemma, there are an even number of odd-degree vertices in M . One of the classic results in combinatorial optimization is that given a complete graph (on an even number of vertices), it is possible to calculate

a perfect matching of minimum total cost in polynomial time. Considering that, we can calculate it for O . If we add the edges of the perfect matching over O in M , we create an Eulerian multigraph H since it is connected and all vertices have even degrees. Finally, we can shortcut the duplicated edges to create a solution of no greater cost corresponding to a Hamiltonian cycle.

Interestingly, more than four decades of research failed to improve upon **Christofides'** $3/2$ factor, until **Karlin et al. [2021]** provided a randomized $(3/2 - \epsilon)$ -approximation for $\epsilon > 10^{-36}$. Their method follows similar principles to **Christofides'** algorithm but uses a randomly chosen tree from a carefully selected random distribution in place of the minimum spanning tree.

It is generally known that finding PTASes is difficult for a lot of problems. In particular, **Arora et al. [1998b]** showed that there is no PTAS for metric TSP, as well as for the Steiner Tree Problem, Maximum Directed Cut Problem (in which we want to find a bipartition of the vertices of the graph such that the cost of the edges crossing the partition is maximum), and Shortest Superstring Problem (in which we want to find a shortest possible string that contains every string in a given set as substrings), unless $\mathcal{P} = \mathcal{NP}$.

Arora [1996] designed a PTAS for **Euclidean TSP**. Their strategy consisted in recursively partitioning the plane (using a randomized variant of the quadtree) such that some $(1 + \epsilon)$ -approximate salesman tour crosses each line of the partition at most $\mathcal{O}(1/\epsilon)$ times. Such a tour is found by dynamic programming. For each line in the partition, the algorithm first “guesses” where the tour crosses this line and the order in which those crossings occur. Then it recurses independently on the two sides of the line. Only $n \log n$ distinct regions are in the partition, where n is the number of nodes in the plane. Furthermore, the “guess” can be fairly coarse, so the algorithm spends only $\mathcal{O}(\log n)^{\mathcal{O}(1/\epsilon)}$ time per region, for a total running time of $n \cdot (\log n)^{\mathcal{O}(1/\epsilon)}$. The result still holds even if the nodes are in $\mathbb{R}^d$, but the running time increases to $\mathcal{O}(n(\log n)^{(\mathcal{O}(\sqrt{dc}))^{d-1}})$.

One of the few practical implementations of a PTAS in the literature was done by **Rodeker et al. [2009]** for **Arora's** algorithm. Their work describes an implementation of the Euclidean TSP that is based on the essential steps of **Arora's** PTAS with some additional heuristics to improve the running time. In general, their results showed that this PTAS can achieve good performance in practice, although due to challenges during implementation, the authors had to make decisions that caused the loss of theoretical guarantees of the solution's quality. This illustrates how challenging implementing a PTAS can be.

Baker [1994] presented a method for obtaining PTASes for a variety of optimization problems in planar graphs, e.g. maximum-weight independent set and minimum-weight vertex cover. This method was generalized by **Demaine and Hajiaghayi [2005]** to derive algorithms for nonlocal problems, such as feedback vertex set and connected dominating

set. They were able to derive approximation schemes for subclasses of minor-excluded graphs that involve turning the input graph into a low treewidth graph. That way, their results apply to graphs that are not planar.

Klein [2008] improved the results from the work of Althöfer et al. [1993] by generating an EPTAS for TSP in planar graphs. To do so, the authors proposed a framework based on the results from Baker [1994] and Demaine and Hajiaghayi [2005], which consists of the following steps.

The first step creates a subgraph H called **spanner**. A spanner H has two properties of interest: the Quasi-optimality property, which means there is a solution of cost $(1 + \epsilon) \text{OPT}$ in H and the Shortness property, which means the cost of H is bounded by a constant times OPT . The second step is to find a set of edges of cost at most $\mathcal{O}(\epsilon \text{OPT})$ contained in H , then proceed to contract those edges. According to Demaine et al. [2010], the resulting graph of this process has bounded treewidth. The third step uses a dynamic programming approach on this bounded treewidth graph to obtain an optimal solution. Finally, the solution is the union of the resulting graph from the dynamic programming with the set of edges that was contracted.

Klein [2008] also showed that this framework could be generalized to obtain PTASes for multiple problems in planar graphs. Bateni et al. [2011] even commented: “Most approximation schemes for planar graph problems use (implicitly or explicitly) the fact that the problem is easy to solve on bounded treewidth graphs”. That fact is the basis for his work on the Steiner Forest Problem (which will be presented later in this chapter).

Rao and Smith [1998] improved on Arora [1996] work by using “light” spanners for Euclidean graphs to obtain an EPTAS for Euclidean TSP. Demaine et al. [2011] proved that any graph excluding a fixed minor can have its edges partitioned into a desired number k of color classes such that contracting the edges in any one color class results in a graph of treewidth linear in k . This result allowed them to generalize Klein’s framework to H-minor-free graphs. Given that, and using the spanners proposed by Grigni and Sissokho [2002], Demaine et al. proposed the first PTAS for TSP in weighted H-minor-free graphs. This result improved upon the early work by Grigni and Sissokho [2002], who designed a QPTAS for the same problem.

Since Grigni and Sissokho’s spanner has weight $\mathcal{O}(\log(n) \text{poly}(1/\epsilon)) \text{OPT}$, Demaine et al.’s PTAS is not efficient. Borradaile et al. [2017] improved on the spanner’s size resulting in an EPTAS for the problem. The authors mentioned that designing a PTAS for the Subset TSP (also known as Steiner Cycle Problem) in H-minor-free graphs is a central open problem of the field (we will present more on this ahead).

Two related problems to the TSP are the **Steiner Cycle Problem** and the **Multiple Steiner Cycle Problem**. These will be presented in the next section.

3.2 Steiner-related problems

We loosely define a “Steiner-related” problem as any problem that aims at making some kind of connection between terminals inside one or more sets of vertices. Traditionally, these connections are made with trees or cycles.

During this section, we will present some relevant results in the literature for a few Steiner-related problems, in particular the Steiner Tree Problem, Steiner Cycle Problem, the Steiner Forest Problem, and the Multiple Steiner Traveling Salesman Problem. At the end of the section, we will present some results for the SMCP.

3.2.1 Steiner Tree Problem

The Steiner Tree Problem (STP) is one of the most studied problems in the combinatorial optimization literature. The problem consists of finding a tree with minimum cost that connects a subset of the vertices, called terminals, in the graph. It was one of the $\mathcal{NP}$ -complete problems presented by Karp [1972].

In their work, Borradaile et al. [2009] drew inspiration from Klein [2008] framework to create a PTAS for the Steiner Tree Problem in planar graphs. The authors built a spanner using a subgraph MG , called Mortar Graph, and a set of **bricks** - subgraphs that are only connected to the rest of the graph via a bounded set of vertices called **portals**. More details of those concepts are given ahead. This spanner would suffice to be applied to Klein’s framework to obtain a PTAS, but it was noted by the authors that such PTAS would yield a time complexity double exponential on $1/\epsilon$.

To improve this result, the authors decomposed MG into a set of subgraphs called **parcels**. This decomposition was done via a breadth-first search in the dual of MG . As a result, each parcel is an embedded planar subgraph of MG . From this, the authors defined a simple graph called **parcel graph** in which each vertex represents a parcel in MG .

The constructed parcel graph has some interesting properties, one of which is that the planar dual of each parcel has a spanning tree with depth limited by a constant. That way, they can compute the optimal Steiner Tree for each parcel and take their union to obtain a solution for the original graph. This new approach resulted in a running time singly exponential in $\text{poly}(1/\epsilon)$.

To use the Mortar Graph to build the spanner, the authors presented and demonstrated a structural theorem that guarantees that the optimal solution found in the Mortar

Graph, and its bricks, is at most $(1 + \epsilon) \text{OPT}$.

3.2.2 Steiner Cycle Problem

The SMCP was also studied in a more restricted variant, where we must cover all terminals with a single cycle. This more restricted problem is known in the literature as **Steiner Cycle Problem**, SCP (it is also known as **Steiner TSP** or **Subset TSP**). It is possible to observe that the SCP is equivalent to the TSP in the scenario where all vertices are terminals.

Cornuéjols et al. [1985] first introduced the SCP (named Steiner TSP by the authors) while studying the problem in its graphical case and investigating its polyhedron in series-parallel graphs.

Salazar González [2003] analyzed the polyhedral structure associated with the Steiner Cycle Problem and introduced two lifting strategies to generate inequalities facet defining based on the TSP polytope.

Steinová [2012] presented a $3/2$ -approximation for the SCP in metric graphs. The authors also showed that there is no constant time approximation for general graphs unless $\mathcal{P} = \mathcal{NP}$. This result implies that the SCP, and by consequence the SMCP, is NPO-hard in general graphs.

Arora et al. [1998a] found a *quasipolynomial-time approximation scheme* for the SCP in planar graphs. To find a $(1 + \epsilon)$ -approximated solution for the problem, the proposed algorithm requires time $\mathcal{O}(n^{\text{poly}(\log n, 1/\epsilon)})$. They made the following conjecture that allows the derivation of a PTAS from this result: “There exists a function $f(\cdot)$ such that given $\epsilon > 0$, a planar graph G with edge-weights, and a subset S of vertices, there exists an edge-induced subgraph G' that $(1 + \epsilon)$ -approximates all distances between nodes in S , and furthermore, G' has total edge weight at most $f(\epsilon)$ times the minimum Steiner tree weight for S .”

Klein [2006] created a PTAS for SCP in planar graphs, the first one proposed for the problem, by following up from Arora et al. [1998a]. They confirmed Arora et al.’s conjecture to be true by showing a constructive proof that implies a polynomial-time algorithm for the construction of the subgraph G' , i.e., the **spanner**.

Klein and Marx [2014] proposed a sub-exponential parameterized algorithm for the problem with time complexity $(2^{\mathcal{O}(\sqrt{k} \log k)} + W) \cdot n^{\mathcal{O}(1)}$, where n is the number of vertices and k is the numbers of terminals, if G is a planar graph with weights that are integers no greater than W . The primary strategy to achieve this result consists of two steps: (1) find a locally optimal solution and (2) use it to guide a dynamic program. The proof of

correctness of the algorithm depends on the treewidth of a graph obtained by combining an optimal solution with a locally optimal solution.

[Le \[2020\]](#) proposed a PTAS for the SCP in H -minor-free graphs, for any fixed graph H , expanding on [Borradaile et al. \[2017\]](#) work. Their main contribution is the concept of a nearly light subset spanner construction based on sparse spanner oracles. They show that spanner oracles with weak sparsity are necessary and sufficient to construct light subset spanners, even for general graphs.

This result is interesting for several reasons. Despite the many advances in meta-algorithms related to PTAS in H -minor-free graphs, a PTAS for SCP on this class of graphs was still unknown. Moreover, there is evidence that a PTAS for this problem is impossible in more general graphs than H -minor-free.

3.2.3 Multiple Steiner Traveling Salesman Problem

The **Multiple Steiner Traveling Salesman Problem with Order Constraints**, or briefly MSTSPO, is a close variant of the Steiner Multicycle Problem. The difference is that the MSTSPO fixes a set of K salesman, each having to visit a set of terminals to form the cycles, and considers that each cycle must visit the terminals in a predefined order.

[Borne et al. \[2013\]](#) formulated the problem motivated by survivability issues in multilayer telecommunication networks as Multiple Steiner TSP with Order Constraints. Their work proposes an integer linear programming formulation for the problem and investigates the associated polytope. They also present new valid inequalities and discuss their facial aspect. Finally, they devised a Branch-and-cut algorithm and presented preliminary computational results.

[Gabrel et al. \[2020\]](#) expanded on [Borne et al.](#)'s work by proposing a few integer programming formulations and both Branch-and-Cut and Branch-and-Price algorithms to solve the problem. They presented extensive computational results, showing the efficiency of the algorithms.

3.2.4 Steiner Forest Problem

Another SMCP related problem is the Steiner Forest Problem (SFP), where instead of using cycles to connect terminal pairs, we restrict ourselves to using only trees. In a similar way that the SMCP generalizes the SCP, the SFP generalizes the STP.

Bateni et al. [2011] proposed a PTAS for the Steiner Forest Problem based on the work of Borradaile et al. [2009]. To do so, the authors presented a clustering algorithm called **Prize Collecting Clustering**, which aims at segmenting the original graph in a set of trees $\{T_1, \dots, T_k\}$ in such a way that the sum of the cost of all trees is at most $(4/\epsilon + 2) \text{OPT}$, where OPT is the cost of an optimal solution for the entire graph, encompassing all terminal sets.

Besides that, considering $\mathcal{D}_i$ as a set of terminals that must be connected, and T_i as the tree that connects the terminals in $\mathcal{D}_i$, the sum of the costs of all the minimal Steiner Forests, connecting only the vertices within each $\mathcal{D}_i$, totals at most $(1 + \epsilon) \text{OPT}$. That can be expressed as $\sum_i \text{OPT}_{\mathcal{D}_i} \leq (1 + \epsilon) \text{OPT}$.

In simpler terms, the union of optimal solutions for each terminal set forms an $1 + \epsilon$ -approximated solution for the input graph. Finally, for each tree T_i obtained by the clustering, the authors applied the framework proposed by Klein [2008], and expanded by Borradaile et al. [2009, 2014].

3.2.5 Steiner Multicycle Problem

Lintzmayer et al. [2020] proposed the first PTAS for the SMCP. The algorithm is a randomized PTAS restricted to the Euclidean plane. In this context, the Euclidean SMCP encompasses a set of terminal pairs distributed across a plane and aim to calculate the line segments that connect the points with the least cost, considering that the same line segment might be crossed more than once.

The algorithm groups the terminal pairs in such a way that pairs that are far away from each other belong to different groups. The authors guarantee that the union of the optimal solution calculated in each group is bounded by a constant times the optimal solution when considering all terminal pairs. Then, for each group of terminal pairs, they create a square called **root dissection square**, which is a bounding box containing all terminal pairs of the group. This square has a cost of at most a constant times the optimal solution cost considering the terminals pairs in the square.

For each square, they run a process of recursive dissection. In this process, each

square is subdivided into four squares of equal size, using vertical and horizontal lines. They select a limited number of points in its border as portals, for each generated square during the partitioning. That way, the solutions generated by the algorithm cross the squares only through the portals. At the end of the process, each square that has not been partitioned is broken into cells, like in a grid.

For each square R , the authors define a memoization table M_R , indexed by valid configurations of R , in other words, valid subpartitions of the cells and portals of R . Considering a valid configuration of R called π , $M_R(\pi)$ is the cost of the minimal solution that is **compatible** with π and **conforms** to R . A solution is compatible with π when its line segments respect the partition of π and a solution conforms with R when it is feasible, connects the necessary terminal pairs, only crosses the borders in R through portals, and makes that crossing a limited number of times.

To write an entry in $M_R(\pi)$, the authors observe all possible configurations of the squares generated by subdividing R , the “children” squares of R , and only consider entries consistent with π from all these configurations. They show that the combined solutions of the four “children” squares generate a solution compatible with π and conform to R .

[Fernandes et al. \[2024\]](#) presented a 3-approximation for SMCP on metric and complete graphs. The algorithm was based on the work done by [Christofides \[1976\]](#) for the metric TSP and consists of the following three basic steps. The first step is to perform a 2-approximation for the Survivable Network Design Problem (SNDP), considering a specific set of parameters which will be detailed next. The authors described the SNDP as follows: given a graph G , a weight function $w : E(G) \rightarrow \mathbb{Q}_+$, and a non-negative integer r_{ij} for each pair of vertices i, j with $i \neq j$, representing a connectivity requirement, the goal is to find a minimum weight subgraph G' of G such that, for every pair of vertices $i, j \in V(G)$ with $i \neq j$, there are at least r_{ij} edge-disjoint paths between i and j in G' .

They also observed that given an SMCP instance, it is easy to define an equivalent SNDP instance by considering $r_{ij} = 2$ for any vertices i and j that belong to the same terminal set and $r_{ij} = 0$ otherwise. That way, the optimum value of the SNDP is also a lower bound on the optimum for the Steiner Multicycle Problem: indeed, an optimal solution for the Steiner Multicycle Problem is a feasible solution for the SNDP with the same cost. Those parameters of the SNDP are used on the first step of the algorithm.

Let T be a set of vertices that has an even number of elements in G . A set J of edges in G is a **T-join** if the set of vertices of G that are incident to an odd number of edges in J is exactly T .

Let G' be the output of the 2-approximation for the SNDP. Consider T to be the set of odd-degree vertices in G' . Thus, the second step of the algorithm consists of calculating in polynomial time a minimum T -join J in G' . Finally, as the third step, we obtain an Eulerian graph H from G' by doubling the edges in J and, by shortcutting an Eulerian tour for each component of H , one obtains a collection C of cycles in G that is

the output of the algorithm.

Borradaile et al. [2014] expanded the works of Borradaile et al. [2009] and Klein [2008] to obtain PTASes in subset connectivity problems in bounded genus graphs. To do that, the authors proposed two methods. The first method is a generalization of the Mortar Graph framework by Klein [2008]. The second applies the strategy proposed in Borradaile et al. [2009], but using the simpler tree decomposition instead of the parcel decomposition.

In this work, we intend to expand on the results found by Lintzmayer et al. [2020] by proposing a PTAS for the SMCP in graphs with bounded genus. To that end, we will use various techniques proposed by Bateni et al. [2011] and Borradaile et al. [2009] for the Steiner Forest and Steiner Tree problems, respectively, such as the Mortar Graph, the Prize Collecting Clustering, and the spanner.

Chapter 4

Computational Experiments

We ran computational experiments in two distinct algorithms on instances for the Steiner Multicycle Problem (SMCP). In particular, we implemented the 3-approximation algorithm proposed by [Fernandes et al. \[2024\]](#), and a relaxed solver to obtain a lower bound to optimal results for the same instances.

It is worth mentioning that we initially tried to run an exact solver to obtain the optimal solutions, but the execution time became prohibitive for this strategy.

All experiments were executed in an Intel Core i5 11400H with a 16Gb 3200MHz RAM. The code was compiled with Clang version 15.0.1 on a Windows 10 machine¹.

The algorithms were implemented in C++ using the Graph library [Lemon \[2023\]](#) and the [Gurobi Optimizer \[2023\]](#).

The linear relaxation was implemented using [Pereira et al. \[2018\]](#) formulation. From [Pereira et al.](#), given an instance (G, c, T) of the SMCP, consider a function $f : 2^V \rightarrow \{0, 1\}$ such that for each non-empty set $S \subset V$ we have $f(S) = 1$ if, and only if, there is some $T_a \in \mathcal{T}$ such that $S \cap T_a \neq \emptyset$ and $T_a \not\subseteq S$, i.e., S is a cut that separates terminals in T_a . They use $i \in T$ to denote a vertex in a terminal. The SMCP can be formulated as:

$$\text{minimize} \quad \sum_{e \in E} c_e x_e \quad (4.1)$$

$$\text{subject to} \quad \sum_{e \in \delta(i)} x_e = 2 \quad \forall i \in \mathcal{T} \quad (4.2)$$

$$\sum_{e \in \delta(S)} x_e \geq 2f(S) \quad \emptyset \neq S \subset V \quad (4.3)$$

$$x_e \in \{0, 1, 2\} \quad e \in E(G) \quad (4.4)$$

Where $\delta(S)$ denotes the set of edges having exactly one endpoint in S . The variable x_e indicates if an edge is used in the solution, Constraint (4.2) assures that exactly one cycle covers each terminal, Constraint (4.3) assures that vertices belonging to a terminal set $T_a \in \mathcal{T}$ are connected, and Constraint (4.4) allows each edge to be used at most twice.

By relaxing the integrality Constraint (4.4) we obtain the linear program used in the implementation. Note that the number of constraints (4.3) is exponential. However,

¹You can see the code at: <https://github.com/RaulWCosta/steiner-multicycle-problem-3-apx>

we managed to compute the relaxed LP in polynomial time by using Gomory-Hu trees to solve maximum flow problems between each pair of vertices in the same terminal set T_a . This is the same strategy employed by [Pereira et al. \[2018\]](#) on their implementations.

4.1 3-approximation Algorithm

As presented in Section 3.2.5, [Fernandes et al. \[2024\]](#) proposed the following algorithm to obtain a 3-approximation for SMCP.

Algorithm 1 SMCP 3-approximation

Input: A complete graph G , a weight function $w : E(G) \rightarrow \mathbb{Q}_+$ satisfying the triangle inequality, and a partition $\mathcal{T} = \{T_1, \dots, T_k\}$ of $V(G)$

Output: A collection $\mathcal{C}$ of cycles that respects $\mathcal{T}$

- 1: $r_{i,j} \leftarrow 2$ for every $i, j \in T_a$ for some $1 \leq a \leq k$
 - 2: $r_{i,j} \leftarrow 0$ for every $i \in T_a$ and $j \in T_b$ for some $1 \leq a < b \leq k$.
 - 3: $G' \leftarrow 2\text{ApproxSND}(G, w, r)$
 - 4: Let T be the set of odd degree vertices in G'
 - 5: Let w' be the restriction of w to the edges in G'
 - 6: $J \leftarrow \text{MinimumTJoin}(G', w', T)$
 - 7: $H \leftarrow G' + J$
 - 8: $\mathcal{C} \leftarrow \text{Shortcutting}(H)$
 - 9: return $\mathcal{C}$
-

To calculate the SNDP in line (3), we implemented the 2-approximation proposed by [Jain \[1998\]](#). This algorithm solves the linear relaxation of the problem and rounds the solutions. We also used Gomory-Hu trees to add cuts to the linear relaxation at each iteration.

As mentioned by the authors, Theorem 1 from [Fernandes et al. \[2024\]](#) implies that a minimum weight perfect matching in the graph $G[T]$ weights at most $w(G')/2$. This means that one can exchange line (6) to compute, instead, a minimum weight perfect matching J in $G[T]$. We applied this strategy in our implementation.

The shortcutting in line (8) is done by finding any Eulerian cycle within each component of H . We conjecture that the quality of the 3-approximation could be further improved by adopting a better strategy for shortcutting, at the cost of a potentially greater execution time.

4.2 Instances

As of the time of writing, the only experimental study available for the SMCP (more specifically, its restricted version R-SMCP) was the one presented by [Pereira et al. \[2018\]](#). In their work, they evaluated implementations of their proposed algorithm and a well-known metaheuristic using two instance types. Type 1 is a set of instances from the multi-commodity one-to-one pickup-and-delivery traveling salesman problem (m-PDTSP), and type 2 is a set of randomly generated instances created by them.

[Hernández-Pérez and Salazar-González \[2009\]](#) generated a set of instances to the m-PDTSP. For each instance, they generated $2n - 2$ uniformly random points with coordinates from -500 to 500 , a vertex in position 0 with coordinates $(0, 0)$ and a vertex in position $2n - 1$ also with coordinates $(0, 0)$ (corresponding to Class 3 of [Hernández-Pérez and Salazar-González \[2009\]](#)). Like [Pereira et al. \[2018\]](#), we only consider the vertex distribution of these instances. For each $i \in \{0, \dots, n - 1\}$, we assign a vertex i as a pickup point of an agent and $i + n$ as its corresponding delivery point. The instances have 6, 11, and 16 agents, totaling 210 instances. This set of instances is of the type 1.

[Pereira et al. \[2018\]](#) generated a set of instances having 16, 32, 64, 128, and 256 agents, where vertices correspond to points distributed in a square of dimensions 100000 by 100000. The square is divided into 1×1 , 2×2 , 3×3 , 4×4 , and 5×5 frames, and each pair of pickup and delivery is in the same frame. The space between frames, which we call a wall, has 0%, 10%, 20%, 30%, or 40% of the frame's size. The wall can be seen as a rectangle separating the different frames (see [Figure 4.1](#)). Notice that there is no wall for the instances with division 1×1 . The location of each point is chosen uniformly at random. For each combination of the number of agents, frames, and wall size, three instances were generated with different seeds, therefore, 315 instances of type 2 were generated.

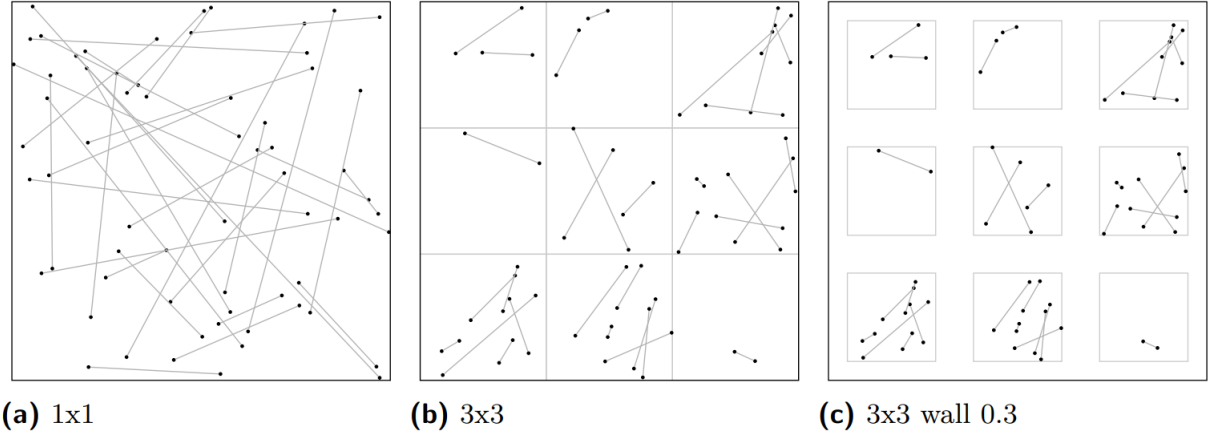


Figure 4.1: Instances randomly generated with 16 agents, in (a) a 1 x 1 frame with no wall, in (b) a 3 x 3 frame with 0% wall, and in (c) a 3 x 3 frame with 30% wall.

4.3 Computational Results

A summary of the experiment results broken by instance group can be seen in Table 4.1. The table shows the gap between the result achieved by the 3-approximation and the linear relaxation, as well as its respective execution times. The first line of the table shows the average results from the type 1 instances. The remaining table subdivides the type 2 instances depending on some properties. The “1 × 1” class contains results with a single frame, while the “2 × 2” class contains the results of the instances with 4 frames, etc. The “W0.x” class contains the results of the instances with wall separation of $(10 \times x)\%$ of the size of the frame (as illustrated in the Figure 4.1 (c)). The last lines of the table separate the instances by the number of “agents” (which is how [Pereira et al.](#) call terminal pairs).

Table 4.1: Summary of results by class.

Classes	num Inst	GAP (%)	APX time (s)	relaxed solver time (s)
m-PDTSP	210	33.80	0.00	0.00
1×1	15	41.06	159.34	0.09
2×2	75	40.67	46.45	0.09
3×3	75	36.46	45.01	0.10
4×4	75	32.89	64.33	0.09
5×5	75	30.77	100.85	0.09
W0.0	75	36.90	145.25	0.09
W0.1	60	37.78	109.89	0.09
W0.2	60	35.36	55.96	0.09
W0.3	60	33.09	7.33	0.09
W0.4	60	32.72	5.91	0.09
rg-016	63	23.76	0.00	0.00
rg-032	63	26.02	0.03	0.00
rg-064	63	34.63	0.56	0.02
rg-128	63	36.74	12.44	0.08
rg-256	63	40.98	330.44	0.35
total average		34.21	197.10	0.29

We can see a positive trend between the number of agents (i.e., terminal pairs) and the solution gap. It is also possible to observe a negative trend between the number of frames and the quality of the solution, indicating that the algorithm does not handle well terminal pairs whose terminals are far from each other.

The relationship between the number of terminals in each instance and the optimality gap of the algorithm can be seen in Figure 4.2. The figure shows that for all instances, the algorithm got significantly closer to the optimality than its 3-approximation guarantee. Although the results, accounting for both execution time and solution quality, were inferior to the heuristic proposed by [Pereira et al. \[2018\]](#).

Although the algorithm tended to perform better on smaller instances, from the 10 worst performance instances (that is, with the largest gap from the linear relaxation), 9 are from the instances of type 1 containing only 6 terminal pairs.

The implementation of the 3-approximation algorithm executes the shortcut step in line (8) by naively finding any Euclidean subgraph contained in each component of the solution; we conjecture that the quality of this algorithm could be improved by applying a more robust shortcutting strategy on the cost of a greater execution time.

In all instances, the time spent to calculate the 2-approximation for the Survivable Network Design Problem accounted for more than 99% of the total execution time of the algorithm. Moreover, the iterative calculation of Gomory-Hu trees, a step during the solving of the SNDP, has a significant impact on the total execution time, especially for bigger instances, as we can see in Table 4.2.

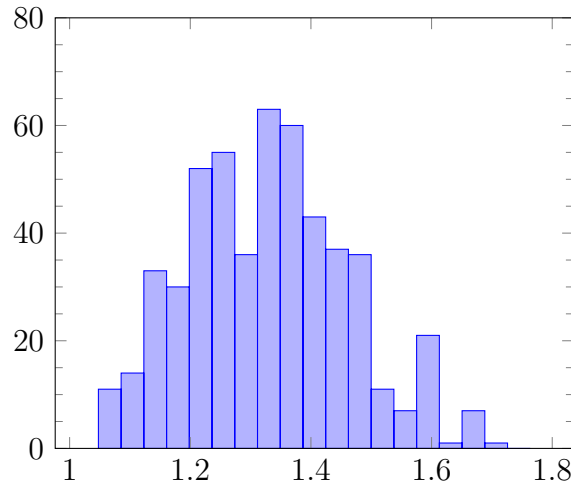


Figure 4.2: Number of instances per optimality gap. An optimality gap of 1.6 means that the solution cost is 60% more than the cost of the relaxation.

Table 4.2: Participation of Gomory-Hu trees calculation on total execution time

num vert	num inst	min (%)	mean (%)	max (%)
12	70	0.00	0.00	0.00
22	70	0.00	0.00	0.00
32	133	0.00	0.00	0.00
64	63	0.00	13.63	40.90
128	63	23.31	39.64	75.80
256	63	30.30	51.86	78.43
512	63	38.68	64.89	91.76

This step might be a good area to improve upon so that the algorithm presents a better execution time in bigger instances.

For practical implementations, a possible alternative would be to use faster heuristics instead of the 2-approximation for the SNDP step (see [Kerivin and Mahjoub \[2005\]](#)). Such heuristics could greatly improve the performance of the algorithm, despite losing theoretical guarantees of the quality of the solutions.

Chapter 5

PTAS for Graphs of Bounded Treewidth

The algorithm receives as input a graph G_{in} with bounded treewidth, and a set $\mathcal{D} = \{T_1, \dots, T_k\}$ of terminal pairs. It begins by defining a polynomial sized subset of all possible solutions. We provide a proof that this subset contains a solution not greater than a constant times OPT. The algorithm then calculates the smallest solution, via a dynamic programming approach, that is contained on this subset and outputs it as result.

We begin the chapter by presenting some tooling, that will be used in the construction of the subset of solutions mentioned before. We then proceed to the dynamic programming algorithm, which is the core of the PTAS.

5.1 Groups

Given a set of vertices X , a set of vertices S (a.k.a. central vertices) such that $S \subseteq X$, and a number r , we define a group $\mathcal{G}_G(X, S, r)$, for some graph G , as the union of S and those vertices of X that are at a distance in G of at most r from some vertex in S .

Lemma 5.1 presented below, states that the distance between any vertex of X and some vertex of S is at most ϵW .

Lemma 5.1. *Let C be a cycle that contains the terminals $X \subseteq V(G)$ and has cost W . For all $0 < \epsilon < 1$, there is a set $S \subseteq X$ of $\mathcal{O}(1 + 1/\epsilon)$ vertices such that $X = \mathcal{G}_G(X, S, \epsilon W)$.*

Proof. We construct S iteratively as follows. First, we choose any vertex in X and add it to S . Then we add the maximum number of vertices from X into S such that the distance between s_i and s_j for $j \in \{1, \dots, i-1\}$ is at least $(\epsilon W)/2$, where s_i and s_j are the i -th and j -th vertices added to S . Let s_t be the last vertex added to S . Note that,

in case $t = 1$, that is s_1 is the only vertex added to S , all the vertices of X are within a maximum distance of $(t\epsilon W)/2$ from vertex s_1 , thus the result holds.

Considering that $t > 1$, the shortest closed walk between the vertices in $S = \{s_1, \dots, s_t\}$, not necessarily in order, has a cost of at least $(t\epsilon W)/2$. Thus $t \leq 2/\epsilon$ is valid. So we conclude that all vertices of X are at distance at most ϵW from $S = \{s_1, \dots, s_t\}$ with $t = 2/\epsilon$. $\square$

Corollary 5.2. *If S_1, S_2, X_1, X_2 are subsets of $V(G)$ and $r_1, r_2 \in \mathbb{R}$, then*

$$\mathcal{G}_G(X_1, S_1, r_1) \cup \mathcal{G}_G(X_2, S_2, r_2) \subseteq \mathcal{G}_G(X_1 \cup X_2, S_1 \cup S_2, \max\{r_1, r_2\})$$

Proof. Let $v \in \mathcal{G}_G(X_i, S_i, r_i)$ with $i \in \{1, 2\}$. The result is valid if $v \in S_i$. Suppose $v \in X_i$, but $v \notin S_i$. Note that $v \in X_1 \cup X_2$, and there is $x \in S_i$ such that the distance between v and x in G is at most r_i . Therefore, v belongs to $\mathcal{G}_G(X_1 \cup X_2, S_1 \cup S_2, \max\{r_1, r_2\})$. $\square$

5.2 Notation and definitions

Let $(T, \mathcal{B})$ denote a nice tree decomposition of G with treewidth κ . Let I be the set of nodes of T , and let $\mathcal{B} = \{B_i : i \in I\}$ be the set of bags of the decomposition.

For a given bag B_i , we denote by V_i the set of vertices that belongs to B_i or any of its descendants in T . We denote as **active** in B_i the set of terminals $A_i \subseteq V_i$ such that for each $t \in A_i$ there is a terminal pair $\{t, t'\} \in \mathcal{D}$ with $t' \in V(G) \setminus V_i$.

Let $G_i = G[V_i]$ and let M_i be a subgraph of G_i that contains A_i . We denote as $\pi_i(M_i)$ the partition of A_i induced by the components of M_i .

A partition α of a set S can be considered an equivalence relation on S . We use $(x, y) \in \alpha$ to indicate that x and y are in the same class of α , and x^α denotes the class of α that contains the element x .

If M is a subgraph of G and $S \subseteq V(G)$, then M induces a partition α of S , where $(x, y) \in \alpha$ if and only if x and y are in the same component of M (and every $x \in S \setminus V(M)$ forms its own class, so M does not have to be a spanning subgraph). We say that a partition α is **finer** than a partition β if $(x, y) \in \alpha$ implies $(x, y) \in \beta$; in that case we say that β is **coarser** than α . We denote as $\alpha_1 \vee \alpha_2$ the finest partition which is coarser than both α_1 and α_2 .

Let β_i be a partition of B_i for some $i \in I$ and let M_i be a subgraph of G_i . We denote as $M_i + \beta_i$ the graph obtained from M_i by adding a new edge xy for each (x, y)

vertex pair that is in the same class in β_i . Note that $M_i + \beta_i$ is not necessarily a subgraph of G_i .

We define the **top bag** of a vertex v as the bag that contains v and is the closest in T to the root. Given two vertices u and v , we say that $u < v$ when the top bag of u is a descendent of the top bag of v . Note that in a nice tree decomposition, each bag is the top for at most one vertex since a forget node can lose only one vertex when compared to its descendent node. Also note that if there is at least one bag that contains both u and v , then either $u < v$ or $v < u$ is valid.

Let K_i , for $i \in \{1, 2\}$, be a set of vertices that induces a connected component in G . Also consider that K_1 and K_2 are disjoint. We extend the previous concept so that $K_1 < K_2$ if there is a vertex v in K_2 such that $u < v$ for all vertices $u \in K_1$. Recall from the definition of the nice tree decomposition, that for a given bag, at most one vertex can be forgotten; considering that, there is always a vertex $v \in K_1 \cup K_2$ such that all vertices in $K_1 \cup K_2 \setminus \{v\}$ are decedents from.

As a consequence of the construction of the nice tree decomposition, if there is a bag containing vertices of both K_1 and K_2 , then $K_1 < K_2$ or $K_2 < K_1$ is valid.

Finally, the following lemma from [Bateni et al. \[2011\]](#) guarantees some properties we shall use in our dynamic programming algorithm.

Lemma 5.3 ([Bateni et al. \[2011\]](#) Lemma 2.1). *Let G be a graph having no connected component consisting of a single edge. G has a nice tree decomposition of polynomial size with the following two additional properties:*

1. *No introduce node introduces a degree 1 vertex.*
2. *The vertices in a join node have a degree greater than 1.*

5.3 Constructing the Partitions

Let Π_i be a set of partitions of the active vertices A_i , where $i \in I$. Let $\Pi = \{\Pi_i\}_{i \in I}$ be a collection of such partitions.

If some collection M of cycles satisfy $\pi_i(M) \in \Pi_i$ for every $i \in I$ (i.e., the partition of A_i induced by M belongs to Π_i), then we say that M **conforms** to Π . This chapter aims to give an algorithm for bounded treewidth graphs that finds a minimum-cost solution conforming to a certain Π that we will build.

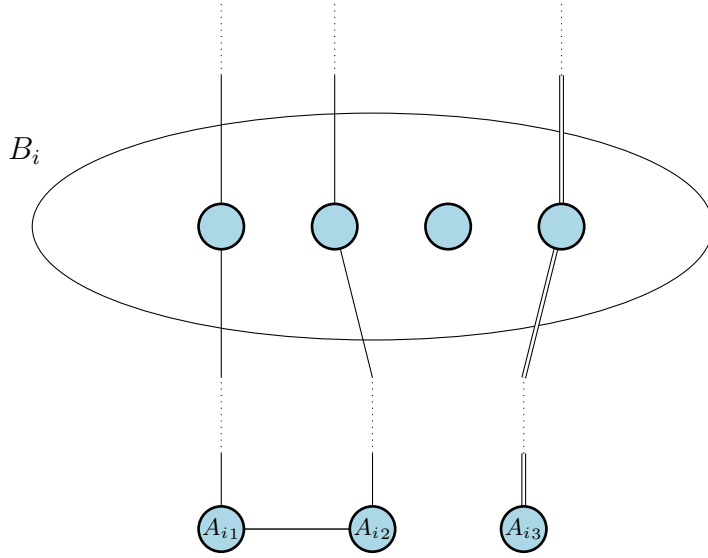


Figure 5.1: View of solution M on bag B_i . This solution induces a partition $\pi_i(M) = \{\{A_{i1}, A_{i2}\}, \{A_{i3}\}\}$ of A_i . If this partition belongs to Π_i , then M conforms to Π_i .

In order to build Π_i , consider $j \in \{1, \dots, \kappa + 1\}$, where κ is the treewidth of G .

Let S_j be a set of vertices of G_i with size $\mathcal{O}((\kappa+1)(1+1/\epsilon))$. Let r_j be a nonnegative real number equal to the distance between any two vertices in G . Think of it as fetching two random vertices from G and assigning the distance between them to r_j .

Each partition π of Π_i is built from a sequence $S = ((S_1, r_1), \dots, (S_p, r_p))$ of $p \leq \kappa + 1$ pairs and a partition ρ of $\{1, \dots, p\}$.

We build the partition π in the following way. Each pair (S_j, r_j) defines a group $R_j = \mathcal{G}_G(A_i, S_j, r_j)$ of A_i where $i \in I$. In order to ensure that the classes of π are disjoint, we define $R'_k := R_k \setminus \bigcup_{l=1}^{k-1} R_l$ for each $k \in \{1, \dots, p\}$.

For each class $P \in \rho$, there is a class in π which corresponds to the union of R'_j for $j \in P$. Precisely, for each $P \in \rho$, let $\bigcup_{j \in P} R'_j$ be a class of π . If π covers all vertices in A_i , then we put the resulting partition of A_i induced by π into Π_i ; otherwise, we ignore π . This process is repeated considering all possible sequences S and all partitions ρ of the sequence's elements.

To show that this process is indeed polynomial, let $n = |V(G)|$ and $y = \mathcal{O}((\kappa + 1)(1+1/\epsilon))$. There are $\binom{n}{y} \leq n^y$ possible sets S_j and, by consequence, $n^y \cdot n^2$ pairs (S_j, r_j) . With that, there are $n^{y+\kappa+1} \cdot n^2$ possible sequences of size $\kappa + 1$ created by pairs (S_j, r_j) and partitioned by ρ . Since we consider all those sequences in order to build Π_i , the size of Π_i is polynomial in $|V(G)|$ considering κ and ϵ constants.

Theorem 5.4. *There is a solution to the SMCP that conforms to Π on graphs of treewidth κ whose cost is no greater than $(1 + 2\kappa\epsilon)$ times the minimum cost.*

Proof. Let G be a graph with treewidth $\kappa-1$, let $\mathcal{D}$ be a set of terminal pairs $\{\{t_1, t'_1\}\}, \dots, \{t_k, t'_k\}\}$,

and let $\epsilon > 0$ be a constant. Let M^* be a minimum-cost collection of cycles concerning G and $\mathcal{D}$.

We will describe a procedure to add edges to M^* in such a way that the resulting collection of cycles M conforms to Π and has a maximum size of $(1 + 2\kappa\epsilon) \cdot c(M^*)$.

Let H^* be the subgraph of G induced by the edges in M^* . Let H be the equivalent to M . Note that, since H is composed of H^* plus some additional edges, H is a super graph of H^* , and the components of H define a partition of the components of H^* .

Initially we set $H = H^*$. Suppose there is a bag B_i whose partition $\pi_i(H)$ of A_i is not in Π_i . Let $K_1 < K_2 < \dots < K_p$ be the components of H^* that intersects B_i , ordered by the relation $<$ (which was presented above). Let ρ be the partition of $\{1, \dots, p\}$ (which corresponds to the components of H^* that intersects B_i) induced by the components of H .

Let $A_{i,j}$ be the subset of A_i connected by K_j . By Lemma 5.1 and Corollary 5.2 there is a set $S_j \subseteq K_j$ of at most $O(1 + 1/\epsilon)$ vertices of K_j such that $A_{i,j} = \mathcal{G}_G(A_{i,j}, S_j, r_j)$, for some $r_j \leq c(K_j)$.

Let π be a partition generated from the sequence $((S_1, r_1), \dots, (S_p, r_p))$ and the partition ρ , as described before. Suppose that $\pi \notin \Pi_i$.

Let R_j and R'_j as defined in Π_i construction (i.e., $R_j = \mathcal{G}_G(A_i, S_j, r_j)$). Let $\rho(j)$ be the class of ρ that contains j . If for all $1 \leq j \leq p$ the vertices of $A_{i,j}$ are contained in $\bigcup_{j' \in \rho(j)} R'_{j'}$ then $\pi \in \Pi_i$. Thus there is a vertex $v \in A_{i,j}$ which is not in $\bigcup_{j' \in \rho(j)} R'_{j'}$. As $v \in R_j$ (since $A_{i,j} \in R_j$) then $v \in R_{j^*}$ also hold for some $K_{j^*} < K_j$ and $j^* \notin \rho(j)$.

The fact that $R_{j^*} = \mathcal{G}_G(A_i, S_{j^*}, r_{j^*})$ contains $v \in A_{i,j}$ means that there is a vertex $u \in S_{j^*}$ such that $\text{dist}_{G_i}(u, v) \leq r_{j^*} \leq \epsilon c(K_{j^*})$. We add to H a shortest path in G connecting u and v . That increases the size of M in at most $2\epsilon c(K_{j^*})$, since each edge of the path may be covered twice, so it guarantees a cycle. This implies that the increased size of H is at most $\epsilon c(K_{j^*})$.

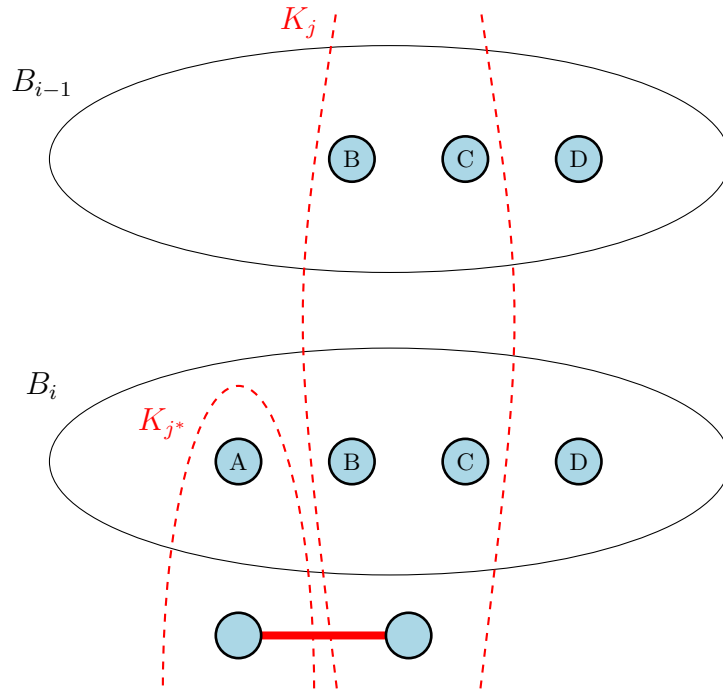


Figure 5.2: Example of connection between K_{j*} and K_j . Note that $K_{j*} < K_j$. The thick red line represents the connection created between both components. This connection consists of the shortest path between both components (it does not necessarily go through edges in B_i).

This increase is due the pair (K_{j*}, K_j) . Notice that this pair of components of H^* are now part of the same component of H and will not be connected again. This process is illustrated in Figure 5.2.

Note that we are charging only to pairs (K, K') having the property that $K < K'$, and there is a bag containing vertices from both K and K' . Observe that if B_i is the topmost bag where vertices from K appear, then these properties imply that a vertex of K' appears in this bag as well. Thus a component K can be the first component of at most κ such pairs (K, K') : since the components are disjoint, the topmost bag containing vertices from K can intersect at most κ other components. We will charge a cost increase of at most $2\epsilon \cdot c(K)$ on the pair (K, K') , thus the total increase is at most $2\kappa\epsilon \cdot c(M^*)$.

Since the modification always extends H , the procedure terminates after a finite number of steps. By the end of this process, each partition $\pi_i(M)$ belongs to the corresponding Π_i , thus the solution M conforms to Π and its cost is at most $(1+2\kappa\epsilon) \cdot c(M^*)$. $\square$

5.4 Conforming Solutions

The primary strategy during the dynamic programming is to use Theorem 5.4 to bound the number of subsolutions we need to evaluate at each bag of the decomposition.

The proof is based on a standard dynamic programming approach. Initially, we perform the following simple transformation on the graph to make the process easier. For each terminal v of G , we create a vertex v' and a zero-weight edge between v and v' , then we replace v by v' as the new terminal. Notice that, after this transformation, all terminals have degree 1, and the size of the minimum Steiner Multicycle stays unchanged.

One well-known fact is that any tree decomposition can be converted into a *nice* tree decomposition of the same width within polynomial time (Cygan et al. [2015], Lemma 7.4).

Consider a nice tree decomposition $(T, \mathcal{B})$ of G . In this decomposition, all terminals are located at leaf nodes, ensuring that join nodes, as per Lemma 5.3, do not introduce any new terminals.

For the dynamic programming, we define a sub-problem for each node $i \in I$. The idea is that the solution of a sub-problem i generates a multiset M_i of edges of G_i in such a way that M_i is equal to the final solution M restricted to G_i .

For simplicity, throughout the proof, we can also observe an edges multiset M as the subgraph induced by its edges. That way, we can talk about components of M , i.e., the components of the subgraph of G induced by the edges in M . Note, however, that $c(M)$ is the sum of the cost of all the edges in the multiset M , considering repetitions.

Due to the possibility that two components of M_i belong to the same component in M , the components of M partition A_i in a coarser way than M_i . Consequently, requiring that $\pi_i(M_i) \in \Pi_i$ during the process is not feasible. To address this issue, we introduce a partition β_i of B_i that captures the partition of the components of M_i induced by the final solution M . Since we do not know M during the process, for each bag B_i , we create a set of solutions considering β as all the possible combinations of the partitions of B_i induced by M_i . Notice that, for each bag B_i , the number of possible combinations of β_i is at most the κ -th Bell number (which can be calculated as $B_\kappa = \sum_{i=0}^{\kappa-1} \binom{\kappa-1}{i} B_{\kappa-1}$), thus a constant number of combinations, considering κ fixed.

Formally, each solution $c = (i, H, \pi, \alpha, \beta, \mu)$ of a subproblem should respect

- (S1) $i \in I$ is a node of T ;
- (S2) H is a multiset of edges that induces a spanning subgraph of $G[B_i]$ (i.e., contains all vertices of $G[B_i]$), in such a way that edges can be crossed multiple times;
- (S3) $\pi \in \Pi_i$ is a partition of A_i ;

(S4) π, β are partitions of B_i , β is coarser than α , and α is coarser than the partition induced by the components of H ;

(S5) μ is a injective mapping from the classes of π to the classes of β .

The item (S5) of the subproblem maps which vertices of B_i belong to the component connected to some vertex of A_i to guarantee that no vertex of A_i becomes “lost” and does not connect to its pair.

A solution $c = (i, H, \pi, \alpha, \beta, \mu)$ of a subproblem is the minimum cost of a multiset M_i with edges of $G[V_i]$, satisfying all of the following requirements:

(C1) $M_i[B_i] = H$ (which implies $B_i \subseteq V(M_i)$);

(C2) α is the partition of B_i induced by M_i ;

(C3) π is the partition of A_i induced by $M_i + \beta$;

(C4) For each descendent d of i , (including $d = i$) the partition of A_d induced by $M_i + \beta$ belongs to Π_d ;

(C5) If there is a terminal pair (x_1, x_2) , with $x_1, x_2 \in V_i$, then they are connected in $M_i + \beta$;

(C6) Every $x \in A_i$ is in the component of $M_i + \beta$ containing $\mu(x^\pi)$.

(C7) Every vertex $v \in M_i$ must have an even degree (accounting for edges duplicity).

With that at hand, we proceed to the main result of this chapter.

Theorem 5.5. *Let G be a graph with treewidth bounded by κ . Let $\epsilon > 0$ be a constant. Let $\mathcal{D}$ be a set of terminal pairs. Let Π be a set of partitions of vertices of G , built as described above. There exists a polynomial-time algorithm to find a multiset of edges M (i.e., a collection of cycles), satisfying the following properties:*

1. *Every vertex $v \in G$ is an endpoint of an even number of edges in M .*
2. *The subgraph induced by M connects all the terminal pairs of $\mathcal{D}$.*
3. *M conforms to Π .*
4. *$c(M) \leq (1 + 2\kappa\epsilon) OPT$.*

Proof. We solve the previously described subproblems by bottom-up dynamic programming, considering each node $i \in I$. We perform different processes for each node type, as detailed below.

Leaf Node i .

If i is a leaf node, the solution's value is trivially 0.

Join Node i with children i_1 and i_2 .

Claim 5.6. *Let $i \in I$ be a join node with children i_1 and i_2 . So, A_{i_1} and A_{i_2} are disjoint and A_i is a subset of the union of $A_{i_1} \cup A_{i_2}$. The value of the subproblem $(i, H, \pi, \alpha, \beta, \mu)$ is:*

$$c(i, H, \pi, \alpha, \beta, \mu) = \min_{(J1), (J2), (J3), (J4)} (c(i_1, H, \pi^1, \alpha^1, \beta, \mu^1) + c(i_2, H, \pi^2, \alpha^2, \beta, \mu^2) - c(H))$$

where the minimum is taken over all tuples satisfying the following properties:

- (J1) $\alpha^1 \vee \alpha^2 = \alpha$;
- (J2) π and π^p are the same on $A_{i_p} \cap A_i$, for each $p \in \{1, 2\}$;
- (J3) For every $p \in \{1, 2\}$ and $v \in A_i \cap A_{i_p}$, $\mu(v^\pi) = \mu^p(v^{\pi^p})$; and
- (J4) If there is a terminal pair (x_1, x_2) with $x_1 \in A_1$ and $x_2 \in A_2$ then $\mu^1(x_1^{\pi^1}) = \mu^2(x_2^{\pi^2})$, in other words, both terminals must be mapped to the same partition of β .

Proof. Let $Q_1 = (i^1, H, \pi^1, \alpha^1, \beta, \mu^1)$ and $Q_2 = (i^2, H, \pi^2, \alpha^2, \beta, \mu^2)$ be subproblems minimizing the right-hand side of (8), and let M_1 and M_2 be optimal solutions of Q_1 and Q_2 , respectively. Let M_i be the union of subgraphs M_1 and M_2 . It is clear that the cost of M_i is exactly the right-hand side of the equation in Claim 5.6 since the common edges of M_1 and M_2 are exactly the edges of H .

We next show that M_i is a solution of P_i , that is, M_i satisfies properties (C1)–(C7).

- (C1) Follows from $M_1[B_i] = M_2[B_i] = M_i[B_i] = H$.
- (C2) Follows from (J1) and from the fact that M_1 and M_2 intersect only in B_i .
- (C3) First consider two vertices $x, y \in A_i^p \cap A_i$. Vertices x and y are connected in $M_i + \beta$ if and only if they are connected in $M_p + \beta$. By (C3) for M_p , this is equivalent to $(x, y) \in \pi_p$, which is further equivalent to $(x, y) \in \pi$ by (J2). Now suppose that $x \in A_i^1 \cap A_i$ and $y \in A_i^2 \cap A_i$. In this case, x and y are connected in $M_i + \beta$ if and only if there is a vertex of B_i reachable from x in $M_1 + \beta$ and from y in $M_2 + \beta$, or in other words, $\mu_1(x_1^\pi) = \mu_2(x_2^\pi)$. By (J3), this is equivalent to $\mu(x^\pi) = \mu(y^\pi)$, or $(x, y) \in \pi$ (as μ is injective).
- (C4) If d is a descendant of i_p with $p \in \{1, 2\}$, then the statement follows using that (C4) holds for solution M_p of P_p and the fact that for every descendant d of i_p , $M_p + \beta$ and $M_i + \beta$ induce the same partition of A_d . For $d = i$, the statement follows from the previous paragraph, that is, from the fact that $M_i + \beta$ induces partition $\pi \in \Pi_i$ of A_i .

- (C5) Consider a pair (x_1, x_2) . If $x_1, x_2 \in V_{i_p}$, then the statement follows from (C5) on M_p . Suppose now that $x_1 \in V_{i_1}$ and $x_2 \in V_{i_2}$; in this case, we have $x_1 \in A_{i_1}$ and $x_2 \in A_{i_2}$. By (C6) on M_1 and M_2 , x_p is connected to $\mu^p(x_p^{\pi^p})$ in $M_p + \beta$. By (J4), we have $\mu^1(x_1^{\pi^1}) = \mu^2(x_2^{\pi^2})$, hence x_1 and x_2 are connected to the same class of β in $M_i + \beta$.
- (C6) Consider an $x \in A_i$ that is in A_{i_p} . By condition (C6) on M_p , we have that x is connected in $M_p + \beta$ (and hence in $M_i + \beta$) to $\mu^p(v_p^\pi)$, which equals $\mu(v^\pi)$ by (J3).
- (C7) Also follows from $M_1[B_i] = M_2[B_i] = M_i[B_i] = H$ since no edges were removed from or added to the children subproblems.

We now proceed to show the other inequality, that is, the left-hand side is at least the right-hand side. Let M_i be a solution of subproblem $(i, H, \pi, \alpha, \beta, \mu)$ and let M_p be the subgraph of M_i induced by V_{i_p} . To prove the inequality, we need to show three things. First, we have to define two tuples $(i_1, H, \pi^1, \alpha^1, \beta, \mu^1)$ and $(i_2, H, \pi^2, \alpha^2, \beta, \mu^2)$ that are subproblems, that is, they satisfy (S1)–(S5). Second, we need to show that (J1)–(J4) hold for these subproblems. Third, we need to show that M_1 and M_2 are solutions for these subproblems (i.e., respects (C1)–(C7)). Hence, they can give an upper bound on the right-hand side that matches the cost of M_i .

Let α^p be the partition of B_i induced by the components of M_p ; as M_1 and M_2 intersect only in B_i , we have $\alpha = \alpha^1 \vee \alpha^2$, ensuring (J1). Since β is coarser than α , it is coarser than both α^1 and α^2 . Let π^p be the partition of A_{i_p} defined by $M_i + \beta$; we have $\pi^p \in \Pi_{i_p}$ by (C4) for M_i . Furthermore, by (C3) for M_i , π is the partition of A_i induced by $M_i + \beta$, hence it is clear that π and π^p are the same on $A_{i_p} \cap A_i$, so (J2) holds. This also means that $M_i + \beta$ (or equivalently, $M_p + \beta$) connects a class of π^p to exactly one class of β ; let μ^p be the corresponding mapping from the classes of π^p to β . Now (J4) is immediate.

It is clear that the tuple $(i_p, H, \pi^p, \alpha^p, \beta, \mu^p)$ satisfies (S1)–(S5), since it is a subproblem. We need to show that M_p is a solution of the subproblem $(i_p, H, \pi^p, \alpha^p, \beta, \mu^p)$. As the edges of H are shared by M_1 , and M_2 , it will follow that the right-hand side is not greater than the left-hand side.

- (C1) Obvious from the definition of M_1 and M_2 .
- (C2) Follows from the definition of α^p .
- (C3) Follows from the definition of π^p , and from the fact that $M_i + \beta$ and $M_p + \beta$ induce the same partition on A_{i_p} .
- (C4) Follows from (C4) on M_i and from the fact that $M_i + \beta$ and $M_p + \beta$ induces the same partition on A_d .

(C5) Suppose that $x_1, x_2 \in V_{i_p}$. Then, by (C5) for M_i , x_1 and x_2 are connected in $M_i + \beta$.

(C6) Follows from the definition of μ^p .

(C7) Follows from $M_1[B_i] = M_2[B_i] = M_i[B_i] = H$.

□

Introduction Node i of vertex v .

Claim 5.7. *Let j be the child of i . Since v is not a terminal, $A_i = A_j$. Let M' be a subgraph of $G[V_j]$, let M_S be the subgraph obtained from M' by adding the vertex v to M' , making v adjacent to $S \subseteq B_j$. To guarantee (C7), we need to ensure that all vertices in $v \cup S$ have even degree, so we get the connection of the least cost between v and each $s \in S$ which guarantees such property.*

Since $|S| \leq \kappa$, this can be calculated in constant time. Let R be the multiset of added edges (possibly with repetition). Note that, for a given vertex $s \in S$, the edge vs needs to be crossed at most twice for the solution to be valid.

If α' is the partition of B_j induced by the components of M' , then we define the partition $\alpha'[v, S]$ of B_i to be the partition obtained by joining all classes of α' that intersects S and adding v to this new class. It is clear that $\alpha'[v, S]$ is the partition of B_i induced by M_S .

The value of the subproblem is:

$$c(i, H, \pi, \alpha, \beta, \mu) = \min_{(I1), (I2), (I3)} c(j, H[B_j], \pi, \alpha', \beta', \mu') + \sum_{e \in R} c(e),$$

Where the minimum is taken over all tuples satisfying the following conditions:

(I1) $\alpha_i = \alpha_j[v, S]$ where S is the set of neighbors of v in H ;

(I2) β' is β restricted to B_j ;

(I3) For every $x \in A_i$, $\mu_i(x^\pi)$ is the class of β containing $\mu'(x^\pi)$.

Proof. Proof of left $\leq$ right: Let M' be an optimal solution of subproblem $P' = (j, H[B_j], \pi, \alpha', \beta', \mu')$. Let M_i be the graph obtained from M' by adding to it the edges of H incident to v ; it is clear that the cost of M_i is exactly the right-hand side. Let us verify that (C1)–(C7) hold for M_i .

(C1) Immediate.

(C2) Holds because of (I1) and the way $\alpha'[v, S]$ was defined.

(C3)–(C5) Observe that $M_i + \beta$ connects two vertices of V_j if and only if $M' + \beta'$ does. Indeed, if a path in $M_i + \beta$ connects two vertices via vertex v , then the two neighbors x, y of v on the path are in the same class of β as v (using that α and β are coarser than the partition induced by H); hence, (I2) implies that x, y are in the same class of β' as well. In particular, for every descendant d of i , the components of $M_i + \beta$ and the components of $M' + \beta$ give the same partition of A_d .

(C6) Follows from (C6) for M' and from (I3).

(C7) Immediate from the choice of R .

Proof of left $\geq$ right: Let M_i be a solution of subproblem $(i, H, \pi, \alpha, \beta, \mu)$ and let M' be the subgraph of M_i induced by V_j . We define a tuple $(j, H[B_j], \pi, \alpha', \beta', \mu')$ that is a subproblem, show that it satisfies (I1)–(I3) and that M' is a solution of this subproblem.

Let α' be the partition of V_j induced by M' and let β' be the restriction of β on B_j ; these definitions ensure that (I1) and (I2) hold. Let $\mu'(x^\pi) = \mu(x^\pi) \setminus \{v\}$, which is a class of β' ; clearly, this ensures (I3). Note that this is well defined, as it is not possible that $\mu(x^\pi)$ is a class of β consisting of only v : by (C6) for M_i , this would mean that v is the only vertex of B_i reachable from x in M_i . Since v is not a terminal vertex, $v = x$, thus if v is reachable from x , then at least one neighbor of v has to be reachable from x as well.

Let us verify that (S1)–(S5) hold for the tuple $(j, H[B_j], \pi, \alpha', \beta', \mu')$. (S1) and (S2) clearly hold. (S3) follows from the fact that (C4) holds for M_i and $A_i = A_j$. To see that (S4) holds, observe that $(x, y) \in \alpha'$ implies $(x, y) \in \beta'$. (S5) is clear from the definition of μ' .

The difference between the cost of M_i and the cost of M' is exactly $\sum_{e \in R} c(e)$. Thus, to show that the left-hand side is at most the right-hand side, it is sufficient to show that M' is a solution of subproblem $(j, H[B_j], \pi, \alpha', \beta', \mu')$.

(C1)–(C2) Immediate.

(C3)–(C5) As in the other direction, follow from the fact that $M' + \beta'$ induces the same partition of V_j as $M_i + \beta$.

(C6) By the definition of μ' , it is clear that $\mu'(x^\pi)$ is exactly the subset of B_j that is reachable from x in $M_i + \beta$ and hence in $M' + \beta'$.

(C7) Immediate from the choice of R .

□

Forget Node i of vertex v .

Claim 5.8. *Let j be the child of i . Thus $V_i = V_j$ and $A_i = A_j$.*

The value of the subproblem is:

$$c(i, H, \pi, \alpha, \beta, \mu) = \min_{(F1), (F2), (F3), (F4), (F5)} c(j, H', \pi, \alpha', \beta', \mu')$$

Where the minimum is taken over all tuples satisfying the following:

(F1) $H'[B_i] = H$;

(F2) α is the restriction of α' to B_i ;

(F3) β is the restriction of β' to B_i and $(x, v) \in \beta'$ if, and only if, $(x, v) \in \alpha'$;

(F4) For every $x \in A_i$, $\mu_i(x^\pi)$ is the (nonempty) set $\mu^j(x^\pi) \setminus \{v\}$ (which implies that $\mu^j(x^\pi)$ contains at least one vertex of B_i);

(F5) The degree of v in H_i must be even, considering multiple crossings in a single edge.

Proof. Proof of left $\leq$ right:

Let M' be a solution of $(j, H', \pi, \alpha', \beta', \mu')$. We show that M' is a solution of $(i, H, \pi, \alpha, \beta, \mu)$ as well.

(C1) Clear because of (F1).

(C2) Clear because of (F2).

(C3)–(C5) We only need to observe that $M' + \beta$ and $M' + \beta'$ have the same components: since by (F3), $(x, v) \in \beta'$ implies $(x, v) \in \alpha'$, the neighbors of v in $M' + \beta'$ are reachable from v in M' , thus $M' + \beta'$ does not add any further connectivity compared to $M' + \beta$.

(C6) Observe that if $\mu'(x^\pi)$ are the vertices of B_j reachable from x in $M' + \beta'$, then $\mu(x^\pi) = \mu'(x^\pi) \setminus \{v\}$ are the vertices of B_i reachable from x in $M' + \beta'$. We have already seen that $M' + \beta$ and $M' + \beta'$ have the same components, thus the nonempty set $\mu(x^\pi)$ is indeed the subset of B_i reachable from x in $M' + \beta$. Furthermore, by (F3), β is the restriction of β' on B_i , thus if $\mu'(x^\pi)$ is a class of β' , then $\mu(x^\pi)$ is a class of β .

(C7) Clear because of (F5).

Proof of left $\geq$ right:

Let M' be a solution of $(j, H, \pi, \alpha, \beta, \mu)$. We define a tuple $(j, H', \pi, \alpha', \beta', \mu')$ that is a subproblem, we show that (F1)–(F3) hold, and that M' is a solution of this subproblem.

Let us define $H' = M'[B_j]$ and let α' be the partition of B_j induced by the components of M' ; these definitions ensure that (F1) and (F2) hold. We define β' as the partition obtained by extending β to B_j such that v belongs to the class of β that contains a vertex $x \in B_i$ with $(x, v) \in \alpha'$ (as β is coarser than the partition induced by H , there is at most one such class; if there is no such class, then we let $\{v\}$ be a class of β'). It is clear that (F3) holds for this β' . Let us note that $M' + \beta$ and $M' + \beta'$ have the same connected components: if $(x, v) \in \beta'$, then x and v are connected in M' .

Let $\mu'(x^\pi)$ be the subset of B_j reachable from x in $M' + \beta'$ (or equivalently, in $M' + \beta$). It is clear that $\mu(x^\pi) = \mu'(x^\pi) \setminus \{v'\}$ holds, hence (F4) is satisfied.

Let us verify first that (S1)–(S5) hold for $(j, H', \pi, \alpha', \beta', \mu')$. Clearly, (S1) and (S2) hold. (S3) follows from the fact that (S3) holds for $(i, H, \pi, \alpha, \beta, \mu)$ and $A_i = A_j$. To see that (S4) holds, observe that if $x, y \in B_i$, then $(x, y) \in \alpha'$ implies $(x, y) \in \alpha$, which implies $(x, y) \in \beta$, which implies $(x, y) \in \beta'$. Furthermore, if $(x, y) \in \alpha'$, then $(x, y) \in \beta'$ by the definition of β . (S5) is clear from the definition of μ' .

We show that M' is a solution of $(j, H', \pi, \alpha', \beta', \mu')$.

(C1) Clear from the definition of H' .

(C2) Clear from the definition of α' .

(C3)–(C5) Follow from the fact that $M' + \beta$ and $M' + \beta'$ have the same connected components.

(C6) Follows from the definition of μ' .

(C7) Clear because of (F5).

Items (1), (2), and (3) of the theorem can be verified from restrictions (C7), (C5), and (C4), respectively. Item (4) is naturally derived from Theorem 5.4 and the construction of the dynamic programming algorithm.

□

□

Chapter 6

PTAS for Graphs of Bounded Genus

As we shall see, the proposed PTAS for SMCP is inspired by the PTAS for the Steiner Tree Problem due to [Bateni et al. \[2011\]](#). Put briefly, the proposed algorithm for SMCP consists of the following steps:

1. The algorithm receives as input a graph G_{in} with bounded genus, and a set $\mathcal{D} = \{T_1, \dots, T_k\}$ of terminal pairs.
2. It begins by creating a spanner subgraph (i.e., a subgraph whose size is bounded by a constant time OPT and has a valid solution with a cost at most $(1 + \epsilon) \text{OPT}$ of G_{in} . This is done with an auxiliary clustering algorithm provided by Theorem [6.1](#).
3. The edges of the spanner are split into k -disjoint sets, then the edges in the smallest set are contracted. After the contraction, the algorithm applies [Demaine et al.](#)'s Theorem ([Demaine et al. \[2010\]](#) Theorem 1.1) to convert the spanner subgraph to a bounded treewidth graph.
4. The algorithm then applies the PTAS for bounded treewidth graphs presented in Chapter [5](#). This step returns a solution with cost of at most a constant times OPT .
5. Finally, it constructs the final solution, i.e., the PTAS for bounded genus, by leveraging the solution in bounded treewidth and the set of edges contracted in the previous step.

Similarly to the Chapter [5](#), we start this chapter by introducing some tooling that will be used in the proof of the PTAS.

6.1 Prize Collecting Partition

This section aims to prove the Prize Collecting Partition Theorem (Theorem [6.1](#)) presented below, allowing us to break a Steiner Multicycle instance into simpler, smaller

ones. The proof relies upon an algorithm called Prize-Collecting Clustering, or PC-clustering, which will be presented later in this section.

Theorem 6.1. *Given an $\epsilon > 0$, a graph G_{in} , a cost function associated with each edge in $E(G_{in})$, and a set $\mathcal{D}$ of terminal pairs, we can compute in polynomial time a set of disjoint components $\{C_1, \dots, C_k\}$ (i.e., a set of subgraphs of G_{in}) with an associated partition of the set of terminal pairs $\{\mathcal{D}_1, \dots, \mathcal{D}_k\}$ with the following properties:*

1. *All sets of terminal pairs are covered, that is, $\mathcal{D} = \bigcup_{i=1}^k \mathcal{D}_i$;*
2. *All the terminals pairs in $\mathcal{D}_i$ are covered by the component C_i ;*
3. *The sum of the cost of all the components C_i is no more than $(16/\epsilon+4) \text{OPT}_{\mathcal{D}}(G_{in})$;*
4. *The sum of the costs of the minimum collection of cycles for each set of terminal pairs $\mathcal{D}_i$ is no more than $1+\epsilon$ times the cost of a minimum solution of the SMCP in G_{in} ; that is, $\sum_i \text{OPT}_{\mathcal{D}_i}(G_{in}) \leq (1 + \epsilon) \text{OPT}_{\mathcal{D}}(G_{in})$.*

The last condition implies that we can solve the problem in each C_i separately and still guarantee an approximation with a small factor.

We start proving Theorem 6.1 by applying a 4-approximation algorithm in G_{in} (for instance, the algorithm for SMCP provided by [Pereira et al. \[2018\]](#)). Clearly, this construction satisfies the first two properties, since by definition an SMCP solution covers all sets, and for each set $\mathcal{D}_i$ a component in the solution connects all of its terminal pairs. To guarantee the last two properties, we need to employ the **PC-clustering** algorithm, which is presented in Section 6.1.1.

6.1.1 Prize-Collecting Clustering

The PC-Clustering algorithm is used in the proof of the following theorem - which in turn will be used to prove Theorem 6.1. It is worth mentioning that the Theorem 6.2 is based on [Bateni et al. \[2011\]](#) (see Theorem 3.1 (Prize-Collecting Clustering)).

Theorem 6.2. *Let $G = (V, E)$ be a graph with a nonnegative edge cost $c(e)$ for each edge e and a potential ϕ_v for each vertex $v \in V$. There is a polynomial-time algorithm that computes a subgraph Z of G such that:*

1. the cost of Z is at most $4 \sum_{v \in V} \phi_v$, and
2. Z is a spanning subgraph of G , and
3. all vertices in Z have even degree, and
4. for any subgraph L of G , there is a set of vertices Q such that:
 - a) $\sum_{v \in Q} \phi_v$ is at most the cost of L , and
 - b) if two vertices $v_1, v_2 \notin Q$ are connected by L , then they are in the same component of Z .

The statement above might look too abstract at first, but it will get more clear with its proof and application. First let us present a brief overview of the PC-Clustering algorithm, followed by a more complete description.

We start with a graph where vertices have associated potentials (or prizes) and edges have a “crossing cost”. The idea is to agglomerate the vertices into disjoint clusters progressively.

Initially, each vertex induces an individual cluster. The total potential of a cluster equals the sum of the potentials of its internal vertices minus the cost of its internal edges. At each step, clusters can use their potential to “pay” for crossing edges to merge with other clusters. That way, the algorithm iteratively spends the potential of the graph’s vertices until all the potential is spent.

Bateni et al. [2011] make an analogy to edge painting while presenting the algorithm, where each vertex is associated with a color, and its colors are used to paint edges to connect separated clusters; when an edge is fully painted, it merges two clusters.

To summarize, at the beginning of the algorithm, each vertex v of $V(G)$ has a potential ϕ_v that can be spent in clusters that contain v (so this cluster can connect to other clusters to form new bigger clusters). We keep track of how much “potential” the vertex v spent on a cluster S via the variable $\gamma_{S,v}$, where $S \subseteq V(G) : v \in S$. Given a cluster C , we generalize the variable γ as $\gamma_C := \sum_{v \in C} \gamma_{C,v}$ to account for the total potential spent by any vertex in the cluster C .

The algorithm consists of two stages: *growth* and *pruning*. At the growth stage, we aim to find a subgraph F_1 and a corresponding γ . Next, we *prune* (i.e., remove) some of the edges of F_1 to get another subgraph F_2 . Finally, we duplicate all edges in F_2 to obtain M_2 .

Noticeably, the following inequalities must be respected throughout the whole process.

$$\sum_{S \subseteq V(G): e \in \delta(S)} \sum_{v \in S} \gamma_{S,v} \leq c(e) \quad \forall e \in E(G) \quad (6.1)$$

$$\sum_{S \subseteq V(G): S \ni v} \gamma_{S,v} \leq \phi_v \quad \forall v \in V \quad (6.2)$$

$$\gamma_{S,v} \geq 0 \quad \forall S \subseteq V(G), v \in S \quad (6.3)$$

The main goal of the growth stage is to build a spanning subgraph F_1 that partitions the vertices of G . The intuition is to think of ϕ_v as an amount of potential that v can spend in a cluster S , to which it belongs (i.e., $v \in S$), to merge S with other clusters in G . However, that must be done in a way that the total spent to cross an edge is no more than the cost of that edge (Constraint (6.1)), and that vertex v of S does not use more potential than it has available (Constraint (6.2)). The Constraint (6.3) ensures that the value that v contributes to S is never negative.

We begin the growth stage with an empty γ (i.e., $\forall v \in V(G)$ and $\forall S \subseteq V(G)$ we have $\gamma_{v,S} = 0$) and a subgraph F_1 also empty (i.e., has no edges). During this step, we keep track of a set of clusters $\mathcal{C}$ which partitions the vertices of $V(G)$; initially, each cluster is composed of a single vertex. We say that a cluster $C \in \mathcal{C}$ is **active** if $\sum_{C' \subseteq C} \sum_{v \in C'} \gamma_{C',v} < \sum_{v \in C} \phi_v$ (i.e. the cluster C still has vertices with remaining potential). Notice that if a vertex $v \in C_1 \subset C_2$ spent $\gamma_{C_1,v}$ to join C_1 with another cluster to form C_2 , the potential $\gamma_{C_1,v}$ cannot be used in C_2 . We say that a vertex is active if it still has some spending potential.

During the growth process, we iteratively spend potential from vertices to grow all active clusters by a value η - which may vary between iterations. It is important to note that an equal amount of η is spent on each cluster within the same iteration, and for each cluster, each of its active vertices spends the same amount of potential. Thus, it is possible for the amount spent between vertices of different clusters to be different, depending on the number of active vertices in each cluster.

Let us consider a specific cluster C and denote by $\kappa(C)$ as the number of active vertices in C . Each active vertex in C spends $\eta/\kappa(C)$ of its available potential into the cluster C . This implies that a total of η will be spent on C during the iteration. The value of η is the largest possible value that still adheres to the Constraints (6.1), (6.2) and (6.3) for all active clusters.

After each growth iteration, some constraints might reach equality. If Constraint (6.2) becomes tight for some vertices, then those vertices become inactive. If all vertices inside a cluster become inactive, then we say that the cluster itself becomes inactive. In case Constraint (6.1) becomes tight for some edge uv , then the potential spent by two neighboring clusters was enough to cover the cost of the edge. This implies that there are two clusters $C_u \ni u$ and $C_v \ni v$ which we can merge into a new cluster $C = C_u \cup C_v$, by

adding uv to F_1 . We then add the new cluster and remove the two former clusters from $\mathcal{C}$.

Note that after each growth iteration, at least one of the restrictions will meet equality for some set of vertices and/or edges. Thus, after a polynomial number of iterations, either all restrictions are met in equality, or the potential of all vertices is spent.

The pruning stage iteratively removes some edges from F_1 to form F_2 . This process is necessary for proving Lemma 6.5. Let $\mathcal{S}$ be the set containing all sets that were a cluster at some point during the growth stage. It can be easily observed that the clusters in $\mathcal{S}$ are laminar and the maximal clusters are the clusters of $\mathcal{C}$. Note that $F_1[C]$ is connected for each $C \in \mathcal{S}$.

Let $\mathcal{B} \subseteq \mathcal{S}$ be the set of all tight clusters at the end of the growth stage, i.e., for each $S \in \mathcal{B}$ we have $\sum_{S' \subseteq S} \sum_{v \in S'} \gamma_{S',v} = \sum_{v \in S} \phi_v$. At the beginning of the stage, we initialize F_2 as F_1 . While there is a cluster $S \in \mathcal{B}$ such that $F_2 \cap \delta(S) = \{e\}$ (notice that if $F_2 \cap \delta(S)$ has more than one edge we do not prune S), we remove the edge e from F_2 .

At the end of this process, F_2 does not have edges of $\delta(C)$, for each $C \in \mathcal{S}$; in other words, no connected component of F_2 can have two vertices such that one belongs to C and the other does not. Finally, we duplicate the edges of F_2 to form M_2 .

For Algorithm 2, we define $\mathcal{C}(v)$ as the cluster currently containing the vertex $v \in V$, that is, $\mathcal{C}(v) := C$ for any $v \in C \in \mathcal{C}$.

Algorithm 2 PC-Clustering

Input: Graph G , and potentials $\phi_v > 0$.

Output: Subgraph M_2 .

- 1: Let $F_1 \leftarrow (V(G), \emptyset)$.
 - 2: Let $\gamma_{S,v} \leftarrow 0$ for any $S \subseteq V : v \in S$.
 - 3: Let $\mathcal{S} \leftarrow \mathcal{C} \leftarrow \{\{v\} : v \in V(G)\}$.
 - 4: **while** there is an active vertex **do**
 - 5: Let η be the largest possible value such that simultaneously increasing γ_C by η for all active clusters C does not violate Constraints (6.1), (6.2) and (6.3).
 - 6: Let $\gamma_{\mathcal{C}(v),v} \leftarrow \gamma_{\mathcal{C}(v),v} + \frac{\eta}{\kappa(\mathcal{C}(v))}$ for all active vertices v .
 - 7: **if** $\exists e \in E(G)$ that is tight and connects two clusters **then**
 - 8: Pick one such edge $e = \{u, v\}$.
 - 9: Let $F_1 \leftarrow F_1 \cup \{e\}$.
 - 10: Let $C \leftarrow \mathcal{C}(u) \cup \mathcal{C}(v)$.
 - 11: Let $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\} \setminus \{\mathcal{C}(u), \mathcal{C}(v)\}$.
 - 12: Let $\mathcal{S} \leftarrow \mathcal{S} \cup \{C\}$.
 - 13: Let $F_2 \leftarrow F_1$.
 - 14: Let $\mathcal{B} \leftarrow \{S \in \mathcal{S} : \sum_{S' \subseteq S} \sum_{v \in S'} \gamma_{S',v} = \sum_{v \in S} \phi_v\}$.
 - 15: **while** $\exists S \in \mathcal{B}$ such that $F_2 \cap \delta(S) = \{e\}$ for an edge e **do**
 - 16: Let $F_2 \leftarrow F_2 \setminus \{e\}$.
 - 17: Duplicate edges of F_2 to form M_2 .
 - 18: Output M_2 .
-

Lemma 6.3 (Bateni et al. [2011] - Lemma 3.3). *The cost of F_2 is at most $2 \sum_{v \in V} \phi_v$.*

Proof. The PC-Clustering algorithm has a notion of time. At each discrete step of the algorithm (i.e., at each iteration of line 4), one of two events happens: a cluster becomes inactive, or an edge becomes tight. We call the “time” between two event points **epoch**.

Notice that, during each epoch, each cluster is either active or inactive, and each active cluster C increases its γ_C value, while all other γ ’s remain unchanged. At time 0, all (singleton) clusters with a strictly positive penalty are active.

We aim at proving that the cost of F_2 is at most $2 \sum_{S \subseteq V: v \in S} \gamma_{S,v} \leq 2 \sum_{v \in V} \phi_v$, where the inequality follows from Constraint (6.2).

Let t_j be the time at which the j^{th} event point occurs in the growth step. So the j^{th} epoch is the time interval between t_{j-1} to t_j . For each cluster C let $\gamma_C^{(j)}$ be the amount $\gamma_C := \sum_{v \in C} \gamma_{C,v}$ that the cluster grew during epoch j , which is $t_j - t_{j-1}$ if it was active during this epoch, or zero otherwise. Thus, $\gamma_C = \sum_j \gamma_C^{(j)}$.

Since each edge e of F_2 was added at some point by the growth stage when its edge packing Constraint (6.1) became tight, we can exactly apportion the cost $c(e)$ amongst the collection of clusters $\{K : e \in \delta(K)\}$ whose variables “paid for” the edge, and can divide this up further by epoch. In other words, $c(e) = \sum_j \sum_{K: e \in \delta(K)} \gamma_K^{(j)}$. Summing over the epochs yields the desired conclusion.

We will now prove that, for an arbitrary epoch j , the total edge cost of F_2 that is apportioned from epoch j is at most $2 \sum_C \gamma_C^{(j)}$. In other words, the total rate at which all active clusters pay for edges of F_2 is at most twice the available potential of the active clusters during the epoch.

Let $\mathcal{C}_j$ be the set of clusters that existed during epoch j (either active or inactive). Consider the graph F_2 and then collapse each cluster $C \in \mathcal{C}_j$ into a supernode. Let H be the resulting graph. We will continue to refer to each supernode as a “cluster” during the rest of the proof. Let us denote the active and inactive clusters in $\mathcal{C}_j$ by $\mathcal{C}_{act}$ and $\mathcal{C}_{dead}$, respectively.

The edges of F_2 that are being partially paid during epoch j are exactly those edges of H that are incident to an active cluster – and the total amount of these edges that are being paid during epoch j is $(t_j - t_{j-1}) \sum_{C \in \mathcal{C}_{act}} \deg_H(C)$. Since every active cluster grows by exactly $t_j - t_{j-1}$ in epoch j , we have:

$$\sum_C \gamma_C^{(j)} \geq \sum_{C \in \mathcal{C}_j} \gamma_C^{(j)} = (t_j - t_{j-1}) |\mathcal{C}_{act}|$$

Now we show that $\sum_{C \in \mathcal{C}_{act}} \deg_H(C) \leq 2 |\mathcal{C}_{act}|$.

Since each cluster induces a connected subtree of F_1 and H is built from the contraction of edges of F_2 , it does not introduce new cycles, which implies that H is a

forest. Also, all the leaves in H must be active, because otherwise the corresponding cluster would have been pruned into another component during the pruning stage.

With this information about H , it is easy to bound $\sum_{C \in \mathcal{C}_{act}} \deg_H(C)$. The total degree in H is at most $2(|\mathcal{C}_{act}| + |\mathcal{C}_{dead}|)$. Noticing that the degree of dead clusters is at least two, we get $\sum_{C \in \mathcal{C}_{act}} \deg_H(C) \leq 2(|\mathcal{C}_{act}| + |\mathcal{C}_{dead}|) - 2|\mathcal{C}_{dead}| = 2|\mathcal{C}_{act}|$ as desired.

Finally, we obtain the following inequality because $t_j - t_{j-1} \geq 0$:

$$(t_j - t_{j-1}) \sum_{C \in \mathcal{C}_{act}} \deg_H(C) \leq 2(t_j - t_{j-1})|\mathcal{C}_{act}|$$

Since the left-hand side of the inequality is the total amount being paid to edges of F_2 during epoch j , by summing through all epochs, we get:

$$c(F_2) \leq 2 \sum_j \sum_C \gamma_C^{(j)} = 2 \sum_{S \subseteq V: v \in S} \gamma_{S,v} \leq 2 \sum_{v \in V} \phi_v.$$

□

Let M_2 be the graph created by duplicating each edge on F_2 . Notice that all vertices in M_2 have even degrees and, besides that, $c(M_2) \leq 4 \sum_{v \in V} \phi_v$, which is formalized in the following result.

Corollary 6.4. *The cost of the graph M_2 is at most $4 \sum_{v \in V} \phi_v$.*

Proof. The result follows immediately from Lemma 6.3. □

The following lemma due to [Bateni et al. \[2011\]](#) gives a sufficient condition for two vertices that end up in the same component of F_2 .

Lemma 6.5 ([Bateni et al. \[2011\]](#) - Lemma 3.4). *Two vertices u and v of $V(G)$ are connected in F_2 if there exist clusters S, S' both containing u and v such that $\gamma_{S,v} > 0$ and $\gamma_{S',u} > 0$.*

Proof. The growth stage connects u and v since $\gamma_{S,v} > 0$ and $u, v \in S$. Consider the path P connecting u and v in F_1 . All the vertices of P are in S and S' . For the sake of reaching a contradiction, suppose some edges of P are pruned. Let e be the first edge being pruned in P .

Thus, there must be a cluster $C \in \mathcal{B}$ which cuts e (i.e. C only contains one endpoint of the edge e); furthermore, $\delta(C) \cap E(P) = \{e\}$, since e is the first edge pruned from P . As C cuts e , the laminarity of the clusters gives $C \subset S, S'$.

In addition, note that C contains precisely one endpoint of the path P (as opposed to precisely one endpoint of the edge e). This holds because if C contained neither or both endpoints of P , so the cluster C could not cut P at exactly one edge.

Let v the endpoint of P in this cluster. We then have $\sum_{C' \subseteq C} \gamma_{C',v} = \phi_v$ because C is tight. However, as C is a *proper* subset of S , this contradicts with $\gamma_{S,v} > 0$, proving that the supposition is false. The case when C contains u is symmetric. $\square$

As mentioned before, [Bateni et al.](#) associate a color to each vertex so that the vertex “spends” its color to fill the edges. Therefore, we say that a cluster H exhausts a vertex’s v color, when $\sum_{C \subseteq H} \gamma_{C,v} = \phi_v$. This analogy will be mentioned in the lemmas ahead.

Lemma 6.6 ([Bateni et al. \[2011\]](#) Lemma 3.5). *If a subgraph H of G connects two vertices u, v from different components of F_2 , then H exhausts the “color” corresponding to at least one of u and v .*

Proof. Given that H connects u and v , suppose that H exhausts neither u nor v : there is a set S containing u and a set S' containing v such that $\gamma_{S,v} > 0$ and $\gamma_{S',u} > 0$ and $E(H) \cap \delta(S) = E(H) \cap \delta(S') = \emptyset$. Since H connects u and v , this is only possible if u and v are both in S and S' . By Lemma 6.5, this implies that F_2 connects u and v , a contradiction. $\square$

We can relate the cost of a subgraph to the potential value of the colors it exhausts.

Lemma 6.7 ([Bateni et al. \[2011\]](#) Lemma 3.6). *Let Q be the set of colors exhausted by subgraph G' of G . The cost of G' is at least $\sum_{v \in Q} \phi_v$.*

Proof. This is quite intuitive. Recall that vertices have associated “colors” which are used to “color” the edges of G during cluster growth. Consider a segment on edges corresponding to cluster S with color v . At least one edge of G' passes through the cut $(S, \bar{S})$. Thus, a portion of the cost of G' can be charged from $\gamma_{S,v}$. Hence, the total cost of the graph G' is at least as large as the total amount of colors paid for by Q . $\square$

We finally have all tools to prove Theorem 6.2.

Proof of Theorem 6.2. The subgraph Z is the graph M_2 , which corresponds to the forest F_2 with duplicated edges, which already guarantees Condition (3).

Condition (1) is given by Corollary 6.4.

To assert Condition (2), we must observe that, as a first step of the growth stage of the Algorithm 2, we initialize each vertex as an individual cluster.

For Condition (4), let Q be the set of vertices exhausted by L . Conditions (4a) and (4b) are proved by Lemma 6.7 and Lemma 6.6, respectively. $\square$

6.1.2 Proof of PC-Partition Theorem

The proof of the PC-Partition Theorem (Theorem 6.1) induces an algorithm that outputs a set of components of an input graph G_{in} . Such an algorithm is presented in Algorithm 3.

Algorithm 3 PC-Partition

Input: Graph G_{in} , a set of costs associated with the edges in EG_{in} and a set of terminal pairs $\mathcal{D}$

Output: Set of components $\{C_i\}_{i=1}^k$ such that C_i covers the terminals in $\mathcal{D}_i$

- 1: Use the algorithm proposed by [Pereira et al. \[2018\]](#) to find a Steiner Multicycle 4-approximated $M^* = \{C_1^*, \dots, C_k^*\}$ of $\mathcal{D}$.
 - 2: Contract each cycle C_i^* in order to create a new graph G .
 - 3: For every $v \in V(G)$, let ϕ_v be equal to $\frac{1}{\epsilon}$ times the cost of the cycle C_i^* that was contracted into v , and 0 in case there is no such cycle.
 - 4: Let M_2 be the solution produced by the PC-Clustering algorithm on G and ϕ_v .
 - 5: Build M from M_2 by *uncontracting* all cycles C_i^* .
 - 6: Return the set of components $\{C_i\}_{i=1}^k$, with $\mathcal{D}_i := \{(s, t) \in \mathcal{D} : s, t \in V(C_i)\}$.
-

Proof of Theorem 6.1. Let G be the graph resulting from the contraction of the components of the 4-approximation M^* (as shown in Algorithm 3). For each component C of M^* contracted, resulting in a vertex v , we define a potential $\phi_v := \frac{1}{\epsilon} \cdot c(C)$ if v is the result of the contraction of a component of M^* and zero otherwise.

Let Z be the subgraph of G given by Theorem 6.2. Let Z_{in} be the subgraph of G_{in} obtained from Z by uncontracting the components of M^* and adding M^* to Z_{in} . Note that Z_{in} is a spanning subgraph of G_{in} , since Z is a spanning subgraph of G (Condition (2)).

Considering that M^* is a valid solution, and each vertex in Z has an even degree (Condition (3)), we have that Z_{in} is a valid solution for the SMCP as well. Let $\{C_1, \dots, C_k\}$ be the components of Z_{in} , and let $\mathcal{D}_1, \dots, \mathcal{D}_k$ be the set of terminal pairs covered by those components. It is clear that Z_{in} satisfies Conditions (1) and (2).

To prove that Z_{in} satisfies Condition (3), note that the cost of Z_{in} is the cost of M^* (which is at most $4 \cdot \text{OPT}$) plus the cost of Z (which, by Theorem 6.2, is at most $4 \sum_{v \in V} \phi_v = \frac{4}{\epsilon} c(M^*) \leq \frac{16}{\epsilon} \text{OPT}$ by construction).

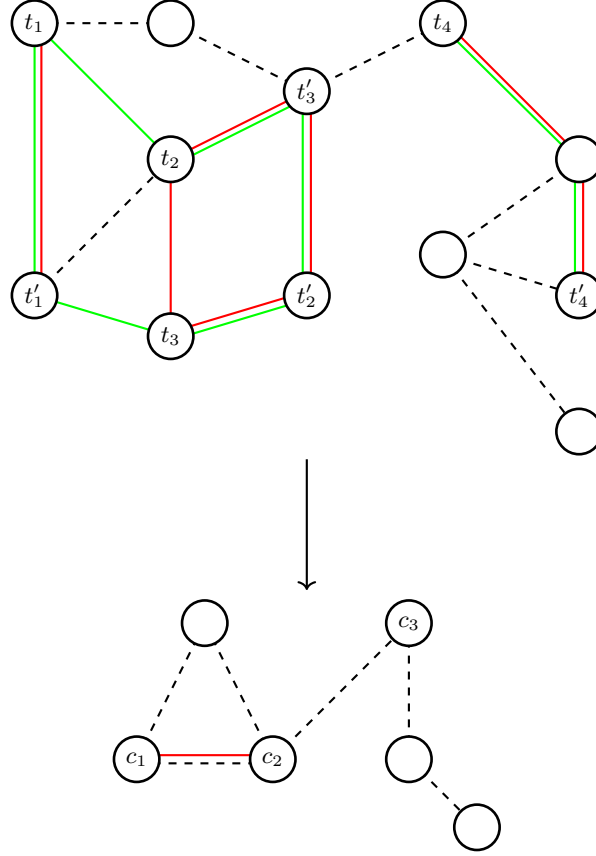


Figure 6.1: In the top figure, the graph G is depicted with the approximated solution M and an optimal solution M^* represented with red and green lines, respectively. The bottom figure illustrates G contracted by M , where each vertex $\{c_1, c_2, c_3\}$ corresponds to a component of M . The graph L consists of the red line and the vertices $\{c_1, c_2, c_3\}$.

Let Z_{in}^* be optimal solution for SMCP on G_{in} , and let L be the subgraph of G corresponding to Z_{in}^* (obtained by contracting the components of M^*), as shown in Figure 6.1.

Let Q be the set of vertices of G given by the Condition (4) of Theorem 6.2. The set Q might contain (or not) vertices from G that are a result of the contraction of components of M^* . It is important to note that there is no need to compute Q and L to execute the algorithm.

Let Q_{in} be the subgraph of G_{in} composed of the components of M^* whose contracted vertices in G belongs to Q . From Condition (4a) of Theorem 6.2, we have $c(Q_{in}) = \epsilon \sum_{v \in Q} \phi_v \leq \epsilon c(L) \leq \epsilon \text{OPT}$.

To show that the last condition holds, for every $\mathcal{D}_i$ (i.e., the set of terminal pairs connected by C_i), we construct a subgraph H_i that satisfies the terminal pairs in $\mathcal{D}_i$. Initially, H_i is empty. For each terminal pair in $\mathcal{D}_i$, if the component K of M^* that satisfies the pair belongs to Q_{in} , then we add K to H_i . Otherwise, we add the component of Z_{in}^* that satisfies the pair into H_i . It is worth mentioning that the algorithm does not actually compute H_i , because that would require to know Z_{in}^* and Q_{in} .

Observe that each component of Q_{in} is used in at most one of the H_i 's: as Q_{in} is a subgraph of Z_{in} , all the terminal pairs satisfied by a component K of Q_{in} belong to the same $\mathcal{D}_i$.

Furthermore, we claim that each component of Z_{in}^* is used in at most one of the H_i 's. Suppose that a component K of Z_{in}^* was used in both H_i and H_j , i.e., K satisfies a terminal pair in $\mathcal{D}_i$ and a terminal pair in $\mathcal{D}_j$. The components of M^* satisfying these two terminal pairs are not in Q_{in} (otherwise we would have put these components into H_i or H_j instead of K), thus they correspond to nodes $v_1, v_2 \notin Q$ in the contracted graph G . Thus L , the contracted version of Z_{in}^* , connects two nodes $v_1, v_2 \notin Q$. In this situation, Condition (4b) of Theorem 6.2 implies that v_1 and v_2 are in the same component of Z and hence the two terminal pairs are satisfied by the same component of Z_{in} . This contradicts that the two terminal pairs are in two different sets $\mathcal{D}_i$ and $\mathcal{D}_j$, since each component C_i of Z_{in} attends all terminal pairs in each $\mathcal{D}_i$.

Since every component of Q_{in} and every component of Z_{in}^* is used by at most one of the H_i 's, we have $\sum_{i=1}^k c(H_i) \leq \text{OPT} + c(Q_{in}) \leq (1 + \epsilon) \text{OPT}$. Each H_i was constructed from components of Q_{in} and Z_{in}^* , which establishes the last condition of the theorem. $\square$

6.2 Spanner

Given a graph G with a genus of at most g , a set of pairs of terminals $\mathcal{D}$, and a constant $\epsilon > 0$, we aim to generate a subgraph H of G with the following properties.

- Quasi-optimality Property: There is a collection of cycles in H that connects all pairs of terminals in $\mathcal{D}$ and has a maximum cost of $(1 + c\epsilon) \text{OPT}_{\mathcal{D}}(G)$ (with c constant), in particular $\text{OPT}_{\mathcal{D}}(H) \leq (1 + c\epsilon) \text{OPT}_{\mathcal{D}}(G)$;
- Shortness Property: The total cost of H is at most $f(\epsilon, g) \cdot \text{OPT}_{\mathcal{D}}(G)$, where $f(\epsilon, g)$ is a constant dependent on ϵ and the genus g of G .

It is worth mentioning that the Quasi-optimality property is also known in the literature as Spanning property.

Theorem 6.8. *Given $\epsilon > 0$ fixed, a bounded-genus graph G_{in} with costs associated with its edges and a set of pairs of terminals $\mathcal{D}$, we can calculate in polynomial-time a spanner H of G_{in} for Steiner Multicycle with respect to $\mathcal{D}$.*

To build a spanner H , we need to apply the PC-Partition technique introduced in Section 6.1. This technique returns a set of components that will serve as the basis for constructing a new structure called *Mortar Graph*.

Throughout this section, for each component C_i obtained from Theorem 6.1, consider T_i as a minimum spanning tree of C_i .

6.2.1 Mortar Graph

The *Mortar Graph* was proposed in Borradaile et al. [2009] for planar graphs and was later expanded in Borradaile et al. [2014] for bounded genus graphs.

It is a grid-like subgraph that spans all terminals and has size $\mathcal{O}(\text{OPT})$. Each face of the Mortar Graph contains a subgraph named **brick**. The crossing between a brick and the rest of the graph can only be done through a limited set of vertices called **portals**, which bounds the number of possible solutions.

The Mortar Graph is built for each T_i obtained from the clustering step (Section 6.1) individually to create a subgraph H_i of G .

6.2.1.1 Cut graph construction

Given a graph G of genus g embedded in a surface of same genus and a connected subgraph T of G we define the process of **cutting** G with T by duplicating every edge of T , along with the vertices, and creating a new face in the embedding of the graph inside the duplicated edges of T . This process is illustrated in Figure 6.2.

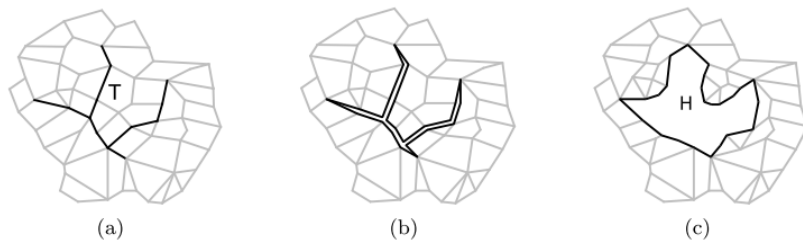


Figure 6.2: Example of cutting process using a subgraph T to form a face H inside the graph. (Borradaile et al. [2014]).

From that, we define a **cut graph** CG as a subgraph of G that when used to cut

G results in a planar graph.

The goal of this section is to find a **cut graph** CG of G that contains all terminals and whose cost is bounded by a constant times OPT . Cutting G using CG results in a planar graph with a cycle σ as the boundary, where σ is twice the cost of CG .

Figure 6.3 illustrates the process of cutting a graph of genus greater than 0. Figure 6.3(a) shows a cut graph drawn on a torus, while Figure 6.3(b) shows the result of cutting the surface along the graph: the shaded area is homeomorphic to a disk, and the light area is the additional face of the planarized surface.

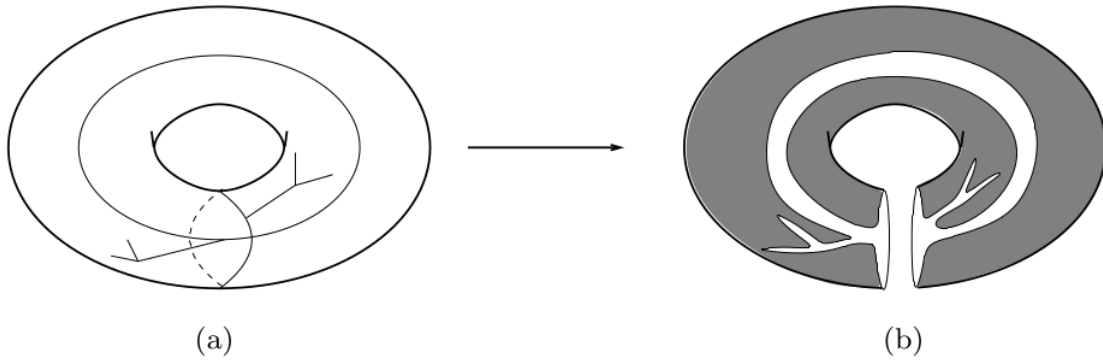


Figure 6.3: Example of a cut in a graph with *genus* greater than zero. (Borradaile et al. [2014]).

Borradaile et al. [2014] observed that, given a planar graph G and a spanning tree T of G , the set of edges $E(G) - E(T)$ induces a spanning tree T^* in the dual graph G^* . Furthermore, if T is a minimum spanning tree of G , then T^* is a maximum spanning tree of G^* . This result was derived by Borradaile et al. from the Lemma 1 of Eppstein et al. [1992].

The result can be generalized for bounded *genus* graphs: if T is a minimum spanning tree of G and T^* is a maximum spanning tree of $G^* - E(T)$, then T^* is a maximum spanning tree of G^* and the size of the set of remaining edges $X := E(G) - E(T) - E(T^*)$ is g , or the Eulerian *genus* of G , obtained by Euler's formula. Eppstein [2003] define such a triple (T, T^*, X) as the *tree-cotree decomposition* of G , and shows that a cut graph can be generated from this decomposition.

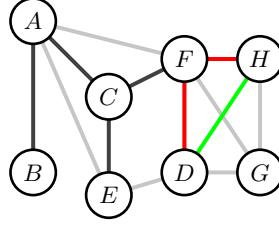


Figure 6.4: Example of $\text{loop}(T, e)$. The tree T is rooted at the vertex F and is represented by the black (non-opaque) edges. The edge e is represented in green and the paths between e and the vertex F are displayed in red. The $\text{loop}(T, e)$ is composed of the union of the red and green edges.

Let T be a spanning tree rooted at a vertex r , and let e be an edge not contained in T . We say that $\text{loop}(T, e)$ is the simple cycle formed by e and the paths between r and both ends of e (exemplified in Figure 6.4). Based on the results from Eppstein, Borradaile et al. [2014] showed in the following result.

Lemma 6.9 (Borradaile et al. [2014], Lemma 1). *Given a tree-cotree decomposition (T, T^*, X) , $\{\text{loop}(T, e) : e \in X\}$, induces a cut graph.*

The construction of the cut graph for our purposes follows from Borradaile et al. [2014], with slight technical changes. We start with T_i , which is a minimum spanning tree of the component C_i calculated in Algorithm 3, and contract it to a vertex r .

We then find a shortest-path tree SPT rooted at r , uncontract r back into T_i and set $T := T_i \cup SPT$, where T is a spanning tree of G . With that, we can then find a spanning tree T^* of $G^* - E(T)$ using the results presented above.

Let $X := E(G) - E(T) - E(T^*)$. As the output cut graph we return $CG := T_i \cup \{\text{loop}(T, e) : e \in X\}$.

We proceed to cut G using CG and duplicate each edge and vertex of CG , thus, generating a cycle σ internal to CG . After this process, let G_1 be the planar graph obtained from G . Finally, we invert the face σ in such a way that σ becomes the new external face of G_1 . From construction and Theorem 6.1, $c(\sigma)$ is at most $c \text{OPT}$ where c is a constant.

Borradaile et al. refers to the algorithm presented above as **Planarize algorithm** and formalizes the result with the following lemma.

Lemma 6.10 (Borradaile et al. [2014], Lemma 2). *The algorithm **Planarize** returns a cut graph CG such that cutting G open along CG results in a planar graph G_p with face f_σ whose facial walk σ*

1. *is a simple cycle,*

2. contains all terminals (some terminals might appear more than once as multiple copies might be created during the cutting process); and
3. has cost $c(\sigma) \leq c \cdot OPT$ for $c > 0$ constant.

6.2.1.2 Strips

To continue with the algorithm, we need to present the following definition. Given $\epsilon > 0$ and a graph G , a path P in G is ϵ -short if in each pair of vertices $(x, y) \in P$ the distance between x and y in P is at most $(1 + \epsilon)$ times the distance x and y in G , in other words, $\text{dist}_P(x, y) \leq (1 + \epsilon)\text{dist}_G(x, y)$.

We proceed to decompose the planar graph G_1 into *strips*. Let x and y be two vertices in σ . Let $\sigma[x, y]$ be the path between x and y in σ in the counterclockwise direction of the *planar embedding* of G_1 into a sphere. If $x = y$, by convention $\sigma[x, y] = \sigma$.

In order to segment G_1 into strips, we find vertices x and y in σ such that $\sigma[x, y]$ is the shortest path of σ which is not ϵ -short in σ . Such a pair of vertices always exists since $\sigma[x, y]$, with $x = y$, is not ϵ -short. Let N be the shortest path between x and y in G_1 . Certainly $c(N) < c(\sigma[x, y])$. We call **strip** the subgraph of G_1 covered by $N \cup \sigma[x, y]$. This process is performed recursively on the subgraph of G_1 covered by $N \cup (\sigma - \sigma[x, y])$. As a result, we have G_1 segmented by strips. Items (a) and (b) of Figure 6.5 illustrate this process.

Lemma 6.11 (Klein [2006], inequality (10)). *The total cost of the boundary of all strips is at most $(\frac{1}{\epsilon} + 1)$ times the cost of σ .*

Klein also showed that the strip decomposition of a planar graph with n vertices can be found in time $\mathcal{O}(n \log n)$.

Given a fixed strip, we denote N as the north-boundary of the strip and S as the south-boundary.

With the graph G_1 decomposed into strips, as illustrated in Item (b) of Figure 6.5, the next step is, for each strip, to calculate the shortest paths, called *columns*. Consider a strip with north and south boundaries, N and S respectively. We select vertices $s_0, s_1, \dots$ in S as follows. We embed the north strip above the south strip, directing S and N from left to right. Let s_0 be the leftmost vertex common to S and N . By convention, the column C_0 is defined as the shortest path between s_0 and N , in this case, an empty path.

For $i \geq 1$, find the first vertex in S (from left to right) such that the cost of the path from s_{i-1} to s_i in S is greater than ϵ times the cost of the shortest path from s_i to N

within the strip, that is, $\text{dist}_S(s_{i-1}, s_i) > \epsilon \cdot \text{dist}_{\text{strip}}(s_i, N)$. Thus, column C_i is defined as the shortest path between s_i and N in the strip, as illustrated by Item (c) of Figure 6.5.

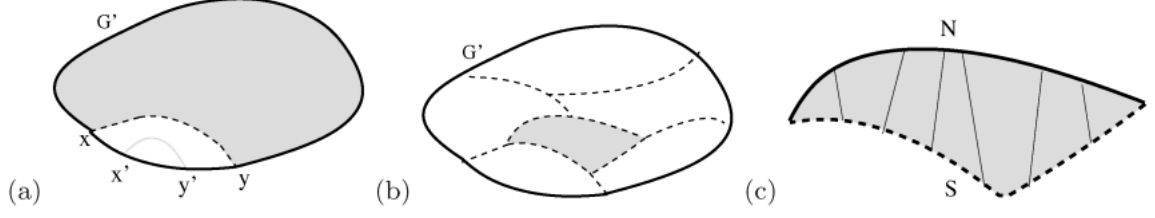


Figure 6.5: Construction of strips in Mortar Graph (Borradaile et al. [2009]).

In Figure 6.5, Item (a) shows the first track is created by a path (dashed) connecting x and y . The distance between every pair of vertices x' and y' , between x and y on the boundary, is well approximated by the distance on the boundary. We use recursion in the shaded area; Item (b) shows how a graph is divided into bands (by the dashed lines). A strip is increased in Item (c). Columns (vertical lines) are taken from the set of shortest paths between the “low”, or south boundary (dashed line) and the “up”, or north boundary (solid line).

Lemma 6.12 (Klein [2006], Lemma 5.2). *The sum of all column costs in a strip is at most $\epsilon^{-1} \cdot c(S)$.*

After having the columns calculated, for each strip, we have a set $C_0, C_1, \dots, C_s$ of columns of the strip.

For $\epsilon > 0$, let $\kappa = \kappa(\epsilon) = 4\epsilon^{-2}(1 + \epsilon^{-1})$. We will use this constant on the following definition and for the following lemmas due to Borradaile et al. [2009].

Let $\mathcal{C}_i = \bigcup_{j=0}^{\kappa-1} C_{i+j\kappa}$ for $i \in \{0, 1, \dots, \kappa - 1\}$.

Let i^* be the index that minimizes $c(\mathcal{C}_i)$. We designate the columns of $\mathcal{C}_{i^*}$ as **super-columns**.

Lemma 6.13 (Borradaile et al. [2009], Lemma 6.5). *The sum of the costs of the super-columns in a strip is at most κ^{-1} times the sum of the costs of the columns in the strip.*

Lemma 6.14 (Borradaile et al. [2009], Lemma 6.6). *The sum of the costs of all the super-columns over all strips is at most $c\epsilon \text{OPT}$, where $\epsilon > 0$ and $c > 0$ depends on ϵ .*

We define as the **Mortar Graph** MG of G the embedded planar subgraph generated by the edges of T_i , the edges of the strips, and the edges of super-columns. This is illustrated in Item (b) of Figure 6.7.

We note two properties derived from the results presented during the construction of MG . First, let Q be the set of all terminals of G , by construction, we have that $Q \subseteq V(MG)$. The second property is presented below.

Lemma 6.15. $c(MG) \leq k\epsilon c(\sigma)$ with $k > 0$ constant.

Proof. The result follows from Theorem 6.1 and Lemmas 6.11 and 6.14. $\square$

6.2.1.3 Bricks

To create a **brick** (illustrated in Figure 6.7(c)), for each face f of MG , we duplicate the border edges and vertices of f . This results in a disconnected induced subgraph of G_1 that is entirely contained within the “external” copy of f ’s border (Figure 6.6(c)). The result of this process can be seen in Figure 6.6.

Figure 6.6(a) illustrates the boundary of a face f of MG as a cycle of edges (thick edges), possibly with repetition (that is, an edge can occur twice at the border). The light edges are those inside f in G . Figure 6.6(b) shows an example of brick B obtained by the process described above. B has boundary ∂B . Figure 6.6(c) shows a brick B contained within the “external” copy of the border of f .

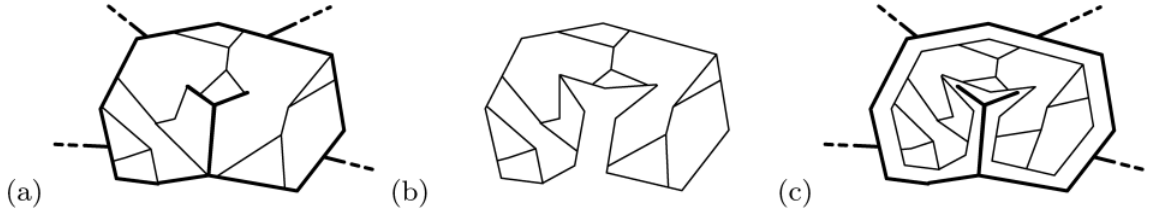


Figure 6.6: Construction of a brick. (Borradaile et al. [2009]).

The boundary ∂B of a brick B is the simple cycle formed by the edges of the boundary. The corresponding face of MG is called **mortar boundary** from B . Each edge of MG occurs at most in the disjoint union of the boundary of the bricks.

Lemma 6.16 (Borradaile et al. [2009], Lemma 6.10). *A mortar boundary B , in counterclockwise order, is the concatenation of four paths W , S , E , N (west, south, east, north) such that:*

1. *The set of edges $B - \partial B$ is nonempty.*

2. Every vertex of $\mathcal{D} \cap B$ is in N or in S .
3. N is 0-short in B , and every proper subpath of S is ϵ -short in S .
4. There exist $k \leq \kappa$ vertices $s_0, s_1, s_2, \dots, s_k$ ordered from west to east along S such that, for any vertex x of $S[s_i, s_{i+1})$, the distance from x to s_i along S is less than ϵ times the distance from x to N in B , that is, $\text{dist}_S(x, s_i) < \epsilon \text{dist}_B(x, N)$.

6.2.1.4 Portals

The connection between a face f of MG and its respective brick B is performed through a subset of vertices of ∂B called **portals**. This connection is made in such a way that, for each portal p of ∂B , an edge with weight 0 is placed between p and the vertex of which it was duplicated in ∂f .

For each brick B , we set some vertices of ∂B as **portals**. We define $\theta = \theta(\epsilon) = 18 \cdot \alpha(\epsilon) \cdot \epsilon^{-2}$. Where $\alpha(\epsilon)$ is a constant that will be defined later.

For portal selection, we use the following greedy algorithm. An initial vertex v_0 of ∂B is selected with uniform probability. Then we define v_1 as the first vertex of ∂B clockwise such that $c(\partial B[v_0, v_1]) > c(\partial B)/\theta$. This process is repeated for v_i in ∂B until $v_0 \in V(\partial B(v_{i-1}, v_i))$.

Using the previous construction, we get the following properties given by [Borradaile et al. \[2009\]](#).

Lemma 6.17 ([Borradaile et al. \[2009\]](#), Lemma 7.1). (*Cover property*): For any vertex $x \in \partial B$, there exists a portal y such that the x -to- y -subpath of ∂B has a maximum cost $c(\partial B)/\theta$.

Lemma 6.18 ([Borradaile et al. \[2009\]](#), Lemma 7.2). (*Cardinality property*): There are at most θ portals in ∂B .

With that, we have a Mortar Graph MG with a set of bricks connected with the face boundaries of MG through portal edges, as illustrated by Item (d) Figure 6.7. This construction is named by [Borradaile et al.](#) as **portal-connected graph**, and is denoted as $\mathcal{B}^+(MG)$.

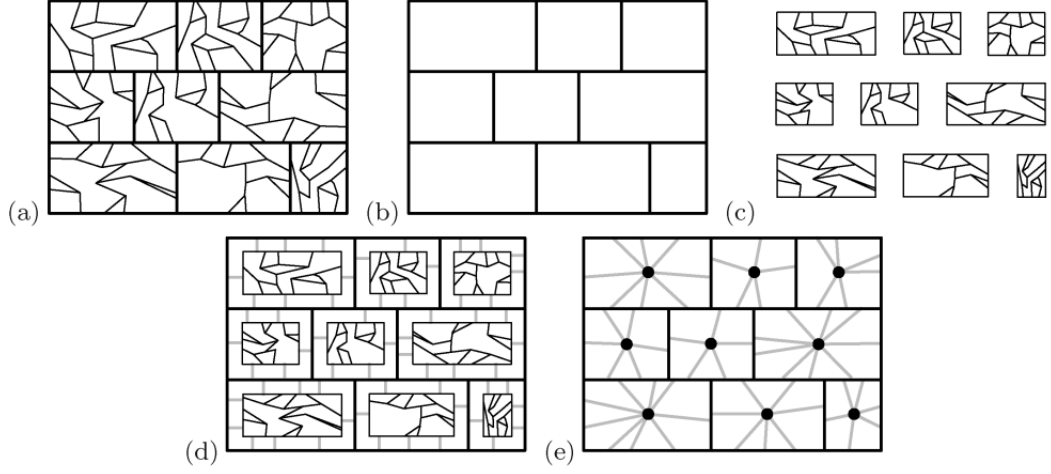


Figure 6.7: Mortar Graph. (Borradaile et al. [2009]).

Figure 6.7 shows in Item (a) the graph G_1 , highlighting the edges of the Mortar Graph. Item (b) shows only the Mortar Graph MG obtained. Item (c) shows the set of bricks corresponding to MG (those will be defined ahead). Item (d) illustrates a portal-connected graph $\mathcal{B}^+(MG)$ (will be presented ahead), where the portal edges are gray. Item (e) shows $\mathcal{B}^+(MG)$ with the bricks contracted into vertices, resulting in $\mathcal{B}^\dagger(\mathcal{B}^+(MG))$.

6.2.2 Spanner construction

Finally, we proceed to prove Theorem 6.8.

We start by building a separate subgraph H_i for each T_i . Let H_i be a Mortar Graph created from T_i . Following that, for each brick B and a selection of its portals $\Pi' \subseteq \Pi$, we add to H_i an optimal Steiner Tree that reaches Π' and uses only edges from the interior of B or its boundary. This can be done in polynomial time in θ using the algorithm proposed in Erickson et al. [1987], since all terminals lie on the infinite face of a planar graph.

Note that for fixed ϵ and g there is at most a constant number of portals, hence a constant number of such Steiner Trees, and the cost of each is at most the cost of the boundary of the brick B .

We will leverage an auxiliary result to prove both spanner properties.

Lemma 6.19. *Let G be a planar embedded graph and let C be a subgraph of G that intersects a ϵ -short path P . There is a subpath of P spanning the vertices of $C \cap P$ whose total cost is $(1 + \epsilon)c(C)$*

Proof. Let P' be the shortest subpath of P that spans all the vertices of $C \cap P$. There is a path Q in C that connects all vertices of $C \cap P$. Since P is ϵ -short $c(P') \leq (1 + \epsilon)c(Q) \leq (1 + \epsilon)c(C)$. $\square$

Lemma 6.20 (Bateni et al. [2011], Lemma 4.1). *For any forest F in a brick B , there exists a forest F' such that*

1. $c(F') \leq (1 + \epsilon)c(F)$;
2. F' crosses the boundary of B at most α times; and
3. any two vertices on N - or S -boundaries of B connected by F are also connected by F'

In Corollary 6.21, we will generalize the result above to a collection of cycles (instead of a forest), but first, we need to introduce the concept of **cleaving**.

Put simply, cleaving is the process of breaking a vertex into two new vertices and adding an artificial, zero-cost edge between them. It can be formalized as: given a vertex v and a bipartition A, B of the edges incident to v , split v into two new vertices v_A and v_B . Then, connect the endpoints of the edges in A , previously connected to v , to v_A , and the edges in B to v_B . Finally, add a zero-cost edge e_{AB} between v_A and v_B . This operation is illustrated in Figure 6.8 (a) and (b).

There can be two types of cleaving:

- **Simplifying Cleaving** (Figure 6.8 (c) and (d)). Let C be a clockwise, non-self-crossing (i.e., planar) non-simple cycle that visits a vertex v twice. Defined a bipartition of the edges incident to v as follows: given the clockwise embedding of the edges incident to v , let A start and end with consecutive edges of C and contain only two edges of C , let B be the remaining edges. Such a bipartition exists because C is non-self-crossing.
- **Lengthening Cleaving** (Figure 6.8 (e) and (f)). Let C be a simple cycle, and let v be a vertex on C with two edges e_A and e_B adjacent to v embedded strictly inside C , and let e'_A and e'_B be consecutive edges of C adjacent to v such that the following bipartition is non-crossing concerning the embedding: A, B is a bipartition of the edges adjacent to v such that $e_A, e'_A \in A$ and $e_B, e'_B \in B$.

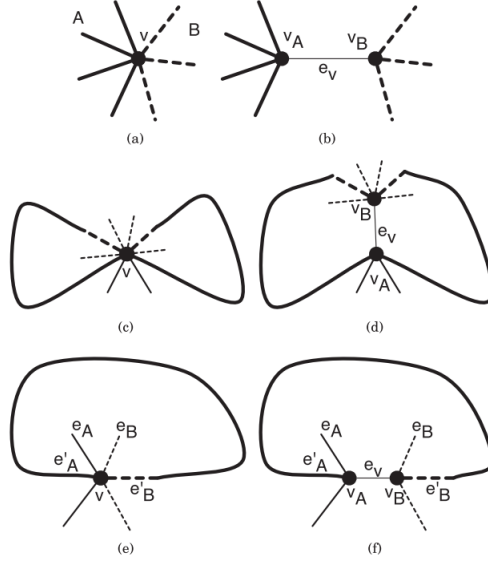


Figure 6.8: Cleavings illustrated (Borradaile and Klein [2016]).

Corollary 6.21. *For any collection of cycles M in a brick B , there is a collection of cycles M' such that*

1. $c(M') \leq (1 + c\epsilon)c(M)$
2. M' crosses the boundary of B at most an α (constant) number of times
3. any two vertices on N - or S -boundaries of B connected by M are also connected by M'

Proof. Let M^* be an optimal solution to the SMCP in G and let MG be a Mortar Graph built from G .

For a given brick B (with W, N, E, S boundaries), let M be the intersection between M^* and B . Note that M might be composed of cycles and trees (e.g., cycle fragments that connect to themselves outside the brick) and each vertex in $M \cap \partial B$.

We begin the construction of M_1 by adding the W and E boundaries of B to M . We know from Lemma 6.14 that the cost of the union of M^* with all super-columns (i.e., the west and east boundaries of all bricks) is at most $(1 + 2c\epsilon) \text{OPT}$.

To streamline the rest of the proof, we perform the simplifying cleaving on non-simple cycles of M_1 until all cycles become simple (also accounting for duplicated edges). We can have two situations for each component C of M_1 :

1. C connects vertices in N or vertices in S
2. C connects vertices from N and S

To tackle case (1), we use Lemma 6.19 to replace the path of C that connects two vertices in N with a subpath of N , which connects the same vertices. Since N is a ϵ -path, the Lemma holds, and the increase in cost is limited by a $(1 + \epsilon)$ factor.

For case (2), we apply the lengthening cleaving to the boundary of brick B . For each vertex v in $M_1 \cap \partial B$ that has a degree greater than one, we perform the lengthening cleaving in v . We repeat this process while there are still multiple edges of the solution embedded in a brick that is incident to a shared boundary vertex. In other words, we have that the intersection of M_1 with the interior of brick B is a subgraph whose joining vertices with ∂B have degree one.

The first property follows, since - as mentioned - in case (1), the increase in cost is limited by a $(1 + \epsilon)$ factor, and in case (2) we only add zero-cost edges.

To get the second and third properties, we can directly apply Lemma 6.20 (Structural properties of Bricks) on M_1 . $\square$

Considering the final spanner H to be the union of all H_i 's, we proceed to prove that H indeed respects both properties of **quasi-optimality** and **shortness**.

Lemma 6.22. (*Shortness property*). *Given a graph G and a subgraph H of G constructed with the process above, the cost of H is at most $f(\epsilon, g) \text{OPT}$ for a certain function $f(\epsilon, g)$.*

Proof. Note that each H_i is constructed from MG_i (which by its turn is built from T_i), a set of portal edges connected to MG_i , and a limited set of Steiner Trees inside the bricks. By Lemma 6.15, $c(MG_i) \leq f(\epsilon, g)c(T_i)$. For each brick B , we add at most 2^θ trees, each with cost no more than $c(\partial B)$. Since each edge of MG_i may appear in at most two bricks, the total cost is bounded by $2^{\theta+1}c(MG_i)$. Therefore, $c(H_i) \leq 2^{\theta+1}c(MG_i) = f(\epsilon, g)c(T_i)$.

By Theorem 6.1, the sum of the cost of all T_i is no more than $(16/\epsilon + 4) \text{OPT}_{\mathcal{D}}(G_{in})$. This implies that the sum of the cost of all H_i is no more than $f(\epsilon, g)(16/\epsilon + 4) \text{OPT}_{\mathcal{D}}(G_{in})$. $\square$

Lemma 6.23. (*Quasi-optimality property*). $\text{OPT}_{\mathcal{D}}(H) \leq (1 + c\epsilon) \text{OPT}_{\mathcal{D}}(G_{in})$ with c and ϵ constants.

Proof. Let $\{C_i\}_{i=1}^k$ be the set of components outputted by Algorithm 3.

From Theorem 6.1, we have that $\sum_i^k \text{OPT}_{\mathcal{D}_i} \leq (1 + \epsilon) \text{OPT}_{\mathcal{D}}$. So we can focus on proving that $\text{OPT}_{\mathcal{D}_i}(H_i) \leq (1 + c\epsilon) \text{OPT}_{\mathcal{D}_i}(G_{in})$.

Consider each H_i as formed in the process above, so each H_i is formed by a Mortar Graph built using T_i (a minimum spanning tree of C_i from Theorem 6.1), the portal edges and the set of Steiner Trees connecting the portals in each Brick.

Let M_i^* be an optimal solution for the SMCP considering $\mathcal{D}_i$, i.e. $c(M_i^*) = \text{OPT}_{\mathcal{D}_i}(G_{in})$. We add the set of all super-columns in H_i to M_i^* to get M_i^1 . Recall that, by Lemma 6.14, the cost of these super-columns is at most $c\epsilon \text{OPT}_{\mathcal{D}_i}(G_{in})$.

Next, we replace the intersection of M_i^1 and each brick with another subgraph having the properties of Corollary 6.21. Let M_i^2 be the new subgraph. The cost of the solution increases to no more than a $1 + c\epsilon$ factor.

From Corollary 6.21, we know that M_i^2 crosses each brick at most α times, so we can ensure that moving these intersection points to the portals adds no more than a constant factor in the cost.

Consider a brick B with boundaries W, N, E, S . Connect each intersection point of the brick to its closest portal. Each connection on a brick B moves by at most $2c(\partial B)/\theta$. Therefore, the total movement of each brick is at most $\alpha 2c(\partial B)/\theta$, which is no more than $2\epsilon^2 c(\partial B)/\gamma(\epsilon, g)$. Hence, the total additional cost for all bricks of H_i is bounded by $4\epsilon^2 c(T_i)$. Let M_i^3 be the resulting subgraph.

Hence the cost of M_i^3 is at most $4\epsilon^2 c(T_i) + c(M_i^2)$. Considering that $c(M_i^2) \leq (1 + \epsilon)c(M_i^1)$ and $c(M_i^1) \leq \epsilon^2 c(T_i) + M_i^*$, we have that $c(M_i^3) \leq 4\epsilon^2 c(T_i) + (1 + \epsilon)c(M_i^*) + \epsilon^2 c(T_i)$.

Let $M' = \bigcup_i^k M_i^3$. Accounting that $\sum_i^k c(T_i) \leq (4/\epsilon + 4) \text{OPT}$ (from Theorem 6.1), it holds that

$$c(M') \leq \sum_i^k (4\epsilon^2 c(T_i) + (1 + \epsilon)c(M_i^*) + \epsilon^2 c(T_i)).$$

Therefore, we can conclude that $c(M') \leq (1 + 21\epsilon + 20\epsilon^2) \text{OPT} = (1 + c'\epsilon) \text{OPT}$. $\square$

6.3 PTAS for Graphs of Bounded Genus

We can now prove one of the main results of this work, which is a PTAS for the Steiner Multicycle Problem on graphs with bounded genus.

First, we state an auxiliary result from Demaine et al. [2010].

Theorem 6.24 (Demaine et al. [2010] Theorem 1.1). *For a fixed genus g , and any integer $k \geq 2$ and for every graph G of Euler genus at most g , the edges of G can be partitioned into k sets such that contracting the edges in any of the sets results in a graph of treewidth $\mathcal{O}((g + 1)^2 k)$. Furthermore, such a partition can be found in $\mathcal{O}((g + 1)^{5/2} n^{3/2} \log n)$ time.*

The proposed PTAS algorithm is presented in Algorithm 4.

Algorithm 4 SMCP-PTAS**Input:** Graph G_{in} of bounded genus g , a set of terminal pairs $\mathcal{D}$, and a constant $1 \geq \epsilon > 0$.**Output:** A collection of cycles satisfying $\mathcal{D}$ whose cost is at most $(1 + c'\epsilon) \text{OPT}$

- 1: $\epsilon' \leftarrow \epsilon/6$
- 2: $k \leftarrow \max\{f(g, \epsilon')/\epsilon', 2\}$; $f(g, \epsilon')$ as in Lemma 6.22
- 3: Construct a spanner H of G_{in}
- 4: Use Theorem 6.24 to partition the edges of H into $E_1, \dots, E_k$
- 5: $j^* \leftarrow \arg \min_{j \in \{1, \dots, k\}} c(E_j)$
- 6: Find a $(1 + k\epsilon)$ collection of cycles (multiset of edges) M^* with respect to $\mathcal{D}_i$ in H/E_{j^*} using Theorem 5.5, and with k constant
- 7: $M \leftarrow M^* \cup E_{j^*}$
- 8: **return** M

Theorem 6.25. *The collection of cycles M produced by Algorithm 4 on input $(G_{in}, \mathcal{D})$ is a feasible solution for the SMCP and has cost at most $(1 + c'\epsilon) \text{OPT}$, with c' and ϵ constants.*

Proof. In the line 3 of Algorithm 4, we construct a spanner H of G_{in} , which respects the properties described in Lemma 6.23 and Lemma 6.22, that is:

- $c(H) \leq f(g, \epsilon) \text{OPT}$, with $f(g, \epsilon)$ being a constant that depends on ϵ and the genus g of G_{in} ;
- $\text{OPT}_{\mathcal{D}}(H) \leq (1 + c\epsilon) \text{OPT}_{\mathcal{D}}(G_{in})$.

Note that Lemma 6.23 implies that there is a feasible solution for the SMCP contained in H .

In line 4, we proceed to partition H using Theorem 6.24. We choose k such that $k = \max(\frac{f(g, \epsilon)}{\epsilon}, 2)$ where $f(g, \epsilon)$ is the constant from Lemma 6.22.

From Lemma 6.24, by splitting H into k edges sets, we have $c(E_{j^*}) \leq \epsilon \text{OPT}$. Moreover, contracting E_{j^*} from H produces a graph of treewidth $\mathcal{O}((g + 1)^2 k)$.

Before contracting H with E_{j^*} , we need to do a small treatment of the terminals contained in E_{j^*} . We create a new vertex for each terminal in E_{j^*} and connect it to the terminal with a new edge of cost zero. This new vertex will serve as a “temporary” terminal in H/E_{j^*} . Note that this process does not increase the cost of the final solution, as all added edges have cost zero.

In line 6 of Algorithm 4, we can find a solution M^* in H/E_{j^*} via Theorem 5.5 and Theorem 6.24. The cost of M^* is at most $(1 + 2\kappa\epsilon) \text{OPT}$, with $\kappa = \kappa(\epsilon) = 4\epsilon^{-2}(1 + \epsilon^{-1})$.

Finally, we join the solution M^* with the edges in E_{j^*} . To guarantee the feasibility of the union as a solution, we duplicate each edge in E_{j^*} . Note that the cost of E_{j^*} accounting for the duplication is at most $2\frac{c(H)}{k}$, which equals $2\frac{f(g, \epsilon)c(G_{in})}{f(g, \epsilon)/\epsilon} = 2\epsilon \cdot c(G_{in})$.

From Theorem 5.5, we have that $c(M^*) \leq (1 + 2\kappa\epsilon \text{OPT})$. So $c(M^* \cup E_{j^*}) \leq (1 + (2\kappa + 2)\epsilon) \text{OPT}$ holds, and by considering a constant $c' = 2\kappa + 2$, we have $c(M) \leq (1 + c'\epsilon) \text{OPT}$.

□

The running time of the algorithm, excluding the bounded treewidth PTAS, is bounded by $\mathcal{O}(n^2 \log n)$. Moreover, the running time of the current procedure for solving bounded treewidth instances is bounded by a polynomial, where k and ϵ appear in the exponent.

Chapter 7

Conclusion

In this work, we first proposed a polynomial-time approximation scheme (PTAS) algorithm to solve the Steiner Multicycle Problem (SMCP) on graphs of bounded *treewidth*. From this result, we also generated a PTAS for the SMCP on graphs of bounded *genus*.

This PTAS was built upon ideas from various sources such as [Borradaile et al. \[2009\]](#), [Borradaile et al. \[2014\]](#), and [Bateni et al. \[2011\]](#), which we adapted to the Steiner Multicycle Problem, particularly to the restricted version R-SMCP. As mentioned in the introduction, we can readily transform instances of SMCP into R-SMCP. The modifications of the algorithm proposed by [Bateni et al.](#) were mainly to extend their definitions and techniques to cycles. In particular, it was necessary to adapt most proofs to guarantee that all vertices in the resulting graphs have even degree.

We also implemented the 3-approximation algorithm proposed by [Fernandes et al. \[2024\]](#) for SMCP, which was tested on the same instances previously used by [Pereira et al. \[2018\]](#). The experimental results showed that the 3-approximation consistently outperformed the theoretical bounds of solution quality. However, the results were still inferior in both quality and running time when compared to the heuristic proposed by [Pereira et al. \[2018\]](#). One significant bottleneck in the algorithm's performance was the Gomory-Hu trees calculation on the 2-approximation for the SNDP, especially for larger instances. We believe further improvements are possible, mainly by employing a more robust algorithm in the short-cutting step and by applying a better strategy to verify the flow between terminal pairs in the 2-approximation for the SNDP step. In particular, for practical implementations, it might be worth considering using a heuristic instead of the 2-approximation in the SNDP step of the algorithm. This heuristic could significantly improve the algorithm's performance, despite losing theoretical guarantees of the quality of the solutions.

For future works, we propose expanding the PTAS for bounded genus graphs algorithm to encompass a more diverse class of graphs, such as H-minor-free graphs. A possible path to achieve this is to use a nearly light subset $(1 + \epsilon)$ -spanner as proposed by [Grigni and Sissokho \[2002\]](#).

Another research direction is to implement the proposed PTAS and carry out computational experiments, as [Tazari and Müller-Hannemann \[2011\]](#) and [Rodeker et al.](#)

[2009] proposed for the Steiner Tree and Euclidean TSP problems, respectively.

References

- Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993. doi: 10.1007/BF02189308. URL <https://doi.org/10.1007/BF02189308>.
- S. Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 2–11, 1996. doi: 10.1109/SFCS.1996.548458.
- Sanjeev Arora, Michelangelo Grigni, David R Karger, Philip N. Klein, and Andrzej Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *ACM-SIAM Symposium on Discrete Algorithms*, 1998a.
- Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, may 1998b. ISSN 0004-5411. doi: 10.1145/278298.278306. URL <https://doi.org/10.1145/278298.278306>.
- Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, jan 1994. ISSN 0004-5411. doi: 10.1145/174644.174650. URL <https://doi.org/10.1145/174644.174650>.
- MohammadHossein Bateni, Mohammad Hajiaghayi, and Dániel Marx. Approximation schemes for Steiner forest on planar graphs and graphs of bounded treewidth. *J. ACM*, 58:21, 10 2011. doi: 10.1145/2027216.2027219.
- J.A. Bondy and U.S.R Murty. *Graph Theory*. Springer Publishing Company, Incorporated, 1st edition, 2008. ISBN 1846289696.
- S. Borne, A.R. Mahjoub, and R. Taktak. A branch-and-cut algorithm for the multiple Steiner TSP with order constraints. *Electronic Notes in Discrete Mathematics*, 41:487–494, 2013. ISSN 1571-0653. doi: <https://doi.org/10.1016/j.endm.2013.05.129>. URL <https://www.sciencedirect.com/science/article/pii/S1571065313001327>.
- Glencora Borradaile and Philip Klein. The two-edge connectivity survivable-network design problem in planar graphs. *ACM Trans. Algorithms*, 12(3), apr 2016. ISSN 1549-6325. doi: 10.1145/2831235. URL <https://doi.org/10.1145/2831235>.

- Glencora Borradaile, Philip Klein, and Claire Mathieu. An $O(n \log(n))$ approximation scheme for Steiner tree in planar graphs. *ACM Trans. Algorithms*, 5(3), jul 2009. ISSN 1549-6325. doi: 10.1145/1541885.1541892. URL <https://doi.org/10.1145/1541885.1541892>.
- Glencora Borradaile, Erik D. Demaine, and Tazari Siamak. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. *Algorithmica*, 68:287–311, 2014. URL <https://doi.org/10.1007/s00453-012-9662-2>.
- Glencora Borradaile, Hung Le, and Christian Wulff-Nilsen. Minor-free graphs have light spanners. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 767–778. IEEE, 2017. doi: 10.1109/FOCS.2017.76. null ; Conference date: 15-10-2017 Through 17-10-2017.
- Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *Operations Research Forum*, 1976.
- G rard Cornu jols, Jean Fonlupt, and Denis Naddef. The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical Programming*, 33(1):1–27, 09 1985. ISSN 1436-4646. doi: 10.1007/BF01582008. URL <https://doi.org/10.1007/BF01582008>.
- Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Cham, 1st edition, 2015. ISBN 9783319212746.
- Erik Demaine and Mohammad Hajiaghayi. Bidimensionality: New connections between FPT algorithms and PTASs. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 590–601, 01 2005. doi: 10.1145/1070432.1070514.
- Erik D. Demaine, MohammadTaghi Hajiaghayi, and Bojan Mohar. Approximation algorithms via contraction decomposition. *Combinatorica*, 30:533–552, 2010.
- Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Contraction decomposition in H-minor-free graphs and algorithmic applications. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC ’11, pages 441–450, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306911. doi: 10.1145/1993636.1993696. URL <https://doi.org/10.1145/1993636.1993696>.
- Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, August 2005. ISBN 3540261826. URL <http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20&path=ASIN/3540261826>.

- David Eppstein. Dynamic generators of topologically embedded graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 599–608, USA, 2003. Society for Industrial and Applied Mathematics. ISBN 0898715385.
- David Eppstein, Giuseppe F Italiano, Roberto Tamassia, Robert E Tarjan, Jeffery Westbrook, and Moti Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *Journal of Algorithms*, 13(1):33–54, 1992. ISSN 0196-6774. doi: [https://doi.org/10.1016/0196-6774\(92\)90004-V](https://doi.org/10.1016/0196-6774(92)90004-V). URL <https://www.sciencedirect.com/science/article/pii/019667749290004V>.
- Ranel E. Erickson, Clyde L. Monma, and Arthur F. Veinott. Send-and-split method for minimum-concave-cost network flows. *Mathematics of Operations Research*, 12(4):634–664, 1987. ISSN 0364765X, 15265471. URL <http://www.jstor.org/stable/3689922>.
- Cristina G. Fernandes, Carla N. Lintzmayer, and Phablo F.S. Moura. Approximations for the steiner multicycle problem. *Theoretical Computer Science*, 1020:114836, 2024. ISSN 0304-3975. doi: <https://doi.org/10.1016/j.tcs.2024.114836>. URL <https://www.sciencedirect.com/science/article/pii/S0304397524004535>.
- Carlos Eduardo Ferreira, CG Fernandes, FK Miyazawa, JAR Soares, JC Pina Jr, KS Guimarães, MH Carvalho, MR Cerioli, P Feofiloff, R Dahab, et al. Uma introdução sucinta a algoritmos de aproximação. *Colóquio Brasileiro de Matemática-IMPA, Rio de Janeiro-RJ*, 2001.
- Virginie Gabrel, A. Ridha Mahjoub, Raouia Taktak, and Eduardo Uchoa. The multiple Steiner TSP with order constraints: complexity and optimization algorithms. *Soft Computing*, 24(23):17957–17968, December 2020. ISSN 1433-7479. doi: 10.1007/s00500-020-05043-y. URL <https://doi.org/10.1007/s00500-020-05043-y>.
- Michelangelo Grigni and Papa Sissokho. Light spanners and approximate TSP in weighted graphs with forbidden minors. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 852–857, USA, 2002. Society for Industrial and Applied Mathematics. ISBN 089871513X.
- Gurobi Optimizer. Gurobi optimizer reference manual, 2023. URL <https://www.gurobi.com>.
- Hipólito Hernández-Pérez and Juan-José Salazar-González. The multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *European Journal of Operational Research*, 196(3):987–995, 2009. ISSN 0377-2217. doi: <https://doi.org/10.1016/j.ejor.2008.05.009>. URL <https://www.sciencedirect.com/science/article/pii/S0377221708004128>.

- Kamal Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21, 05 1998. doi: 10.1007/s004930170004.
- Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric TSP. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 32–45, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380539. doi: 10.1145/3406325.3451009. URL <https://doi.org/10.1145/3406325.3451009>.
- Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972. ISBN 978-1-4684-2001-2. doi: 10.1007/978-1-4684-2001-2_9. URL https://doi.org/10.1007/978-1-4684-2001-2_9.
- Ken-ichi Kawarabayashi, Bojan Mohar, and Bruce Reed. A simpler linear time algorithm for embedding graphs into an arbitrary surface and the genus of graphs of bounded tree-width. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 771–780, 10 2008. doi: 10.1109/FOCS.2008.53.
- Hervé Kerivin and A. Ridha Mahjoub. Design of survivable networks: A survey. *Networks*, 46(1):1–21, 2005. doi: <https://doi.org/10.1002/net.20072>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/net.20072>.
- Philip N. Klein. A subset spanner for planar graphs, with application to subset TSP. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, pages 749–756, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595931341. doi: 10.1145/1132516.1132620. URL <https://doi.org/10.1145/1132516.1132620>.
- Philip N. Klein. A linear-time approximation scheme for TSP in undirected planar graphs with edge-weights. *SIAM Journal on Computing*, 37(6):1926–1952, 2008. doi: 10.1137/060649562. URL <https://doi.org/10.1137/060649562>.
- Philip N. Klein and Dániel Marx. A subexponential parameterized algorithm for subset TSP on planar graphs. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 1812–1830, USA, 2014. Society for Industrial and Applied Mathematics. ISBN 9781611973389.
- Hung Le. A PTAS for subset TSP in minor-free graphs. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, pages 2279–2298, USA, 2020. Society for Industrial and Applied Mathematics.
- Lemon. Library for efficient modeling and optimization in networks, 2023. URL <https://lemon.cs.elte.hu/trac/lemon>.

- Carla N. Lintzmayer, Flávio K. Miyazawa, Phablo F.S. Moura, and Eduardo C. Xavier. Randomized approximation scheme for Steiner multi cycle in the Euclidean plane. *Theoretical Computer Science*, 835:134–155, 2020. ISSN 0304-3975. doi: <https://doi.org/10.1016/j.tcs.2020.06.022>. URL <https://www.sciencedirect.com/science/article/pii/S0304397520303649>.
- Sophie Parragh, Karl Doerner, and Richard Hartl. A survey on pickup and delivery problems: Part i: Transportation between customers and depot. *Journal für Betriebswirtschaft*, 58:21–51, 04 2008. doi: 10.1007/s11301-008-0033-7.
- Vinicius N. G. Pereira, Mário César San Felice, Pedro Henrique D. B. Hokama, and Eduardo C. Xavier. The Steiner Multi Cycle Problem with Applications to a Collaborative Truckload Problem. In Gianlorenzo D’Angelo, editor, *17th International Symposium on Experimental Algorithms (SEA 2018)*, volume 103 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-070-5. doi: 10.4230/LIPIcs.SEA.2018.26. URL <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.SEA.2018.26>.
- Satish B. Rao and Warren D. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 540–550, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 0897919629. doi: 10.1145/276698.276868. URL <https://doi.org/10.1145/276698.276868>.
- Neil Robertson and P.D Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986. ISSN 0196-6774. doi: [https://doi.org/10.1016/0196-6774\(86\)90023-4](https://doi.org/10.1016/0196-6774(86)90023-4). URL <https://www.sciencedirect.com/science/article/pii/0196677486900234>.
- Bárbara Rodeker, María Cifuentes, and Liliana Favre. An empirical analysis of approximation algorithms for Euclidean TSP. pages 190–196, 01 2009.
- Juan José Salazar González. The Steiner cycle polytope. *European Journal of Operational Research*, 147:671–679, 02 2003. doi: 10.1016/S0377-2217(02)00359-4.
- Yinglei Song and Menghong Yu. On the treewidths of graphs of bounded degree. *PLOS ONE*, 10:e0120880, 04 2015. doi: 10.1371/journal.pone.0120880.
- Monika Steinová. Approximability of the minimum Steiner cycle problem. *Computing and Informatics*, 28(1), 2012. URL <http://dml.mathdoc.fr/item/cai148>.

- Siamak Tazari and Matthias Müller-Hannemann. Dealing with large hidden constants: Engineering a planar Steiner tree PTAS. *ACM Journal of Experimental Algorithmics*, 16, 05 2011. doi: 10.1145/1963190.2025382.
- Carsten Thomassen. The graph genus problem is NP-complete. *Journal of Algorithms*, 10(4):568–576, 1989. ISSN 0196-6774. doi: [https://doi.org/10.1016/0196-6774\(89\)90006-0](https://doi.org/10.1016/0196-6774(89)90006-0). URL <https://www.sciencedirect.com/science/article/pii/0196677489900060>.
- David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. ISBN 978-0-521-19527-0.