The Matching Augmentation Problem: A $\frac{7}{4}$-Approximation Algorithm

by

Jack Dippel

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Mathematics

in

Combinatorics and Optimization

Waterloo, Ontario, Canada, 2019

© Jack Dippel 2019

Author's declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The results on the MAP problem are based on joint work with J.Cheriyan, F.Grandoni, A.Khan, and V.V.Narayan, that is posted on Arxiv (arXiv:1810.07816 [cs.DS]).

Abstract

We present a $\frac{7}{4}$ approximation algorithm for the matching augmentation problem (MAP): given a multi-graph with edges of cost either zero or one such that the edges of cost zero form a matching, find a 2-edge connected spanning subgraph (2-ECSS) of minimum cost. We first present a series of approximation guarantee preserving reductions, each of which can be performed in polytime. Performing these reductions gives us a restricted collection of MAP instances. We present a $\frac{7}{4}$ approximation algorithm for this restricted set of MAP instances. The algorithm starts with a subgraph which is a min-cost 2-edge cover, contracts its blocks, adds paths to the subgraph to cover all its bridges, and finally adds cycles to the subgraph to connect all its components. We contract any blocks created throughout. The algorithm ends when the subgraph is a single vertex, and we output all the edges we've contracted which form a 2ECSS.

Acknowledgements

I wish to express my gratitude to Professor Joseph Cheriyan for extended discussions and valuable suggestions which have contributed greatly to the improvement of the thesis.

Table of Contents

List of Figures

Chapter 1

Introduction

A basic goal in the area of *survivable network design* is to design real-world networks of low cost that provide connectivity between pre-specified pairs of nodes even after the failure of a few edges/nodes. Many of the problems in this area are NP-hard, and significant efforts have been devoted in the last few decades to the design of approximation algorithms, see [20].

One of the fundamental problems in the area is the minimum-cost 2-edge connected spanning subgraph problem (abbreviated as min-cost 2-ECSS): given a graph together with non-negative costs for the edges, find a 2-edge connected spanning subgraph (abbreviated as 2-ECSS) of minimum cost. This problem is closely related to the famous Traveling Salesman Problem (TSP), and some of the earliest papers in the area of approximation algorithms address the min-cost 2-ECSS problem [5, 6]. In the context of approximation algorithms, this research led to the discovery of algorithmic paradigms such as the *primal-dual method* [8, 20] and the *iterative rounding method* [10, 13], and led to dozens of publications. Under appropriate assumptions, these methods achieve an approximation guarantee of 2 for several key problems in survivable network design, including min-cost 2-ECSS. Unfortunately, these generic methods do not achieve approximation guarantees below 2. Significant research efforts have been devoted to achieving approximation guarantees below 2 for specific problems in the area of survivable network design. For example, building on earlier work, an approximation guarantee of $\frac{4}{3}$ has been achieved for unweighted (*min-size*) 2-ECSS [17], where each edge of the input graph has cost one and the goal is to find a 2-ECSS with the minimum number of edges.

There is an important obstacle beyond unweighted problems, namely, the special case of min-cost 2-ECSS where the (input) edges have cost of zero or one, and the aim is to design an algorithm that achieves an approximation guarantee below 2. This problem is called the *Forest Augmentation Problem* (FAP). In more detail, we are given an undirected graph $G = (V,\ E_0 \cup E_1)$, where each edge in E_0 has cost zero and each edge in E_1 has cost one; the goal is to compute a 2-ECSS $H = (V, F)$ of minimum cost. Intuitively, the zero-edges define some existing network that we wish to augment (with edges

of cost one) such that the augmented network is resilient to the failure of any one edge. Without loss of generality (w.l.o.g.) we may contract each of the 2-edge connected subgraphs formed by the zero-edges, and hence, we may assume that E_0 induces a forest: this motivates the name of the problem.

A key special case of FAP is the *Tree Augmentation Problem* (TAP), where the edges of cost zero form a spanning tree. Nagamochi [15] first obtained an approximation guarantee below 2 for TAP, and since then there have been several advances including recent work, see [1, 3, 4, 9, 12].

We focus on a different special case of FAP called the *matching augmentation problem* (MAP): given a multi-graph with edges of cost either zero or one such that the edges of cost zero form a matching, find a 2-ECSS of minimum cost. Note that loops are not allowed; multi-edges (parallel edges) are allowed. From the view-point of approximation algorithms, MAP is "orthogonal" to TAP in the sense that the forest of zero-cost edges has many connected components in MAP, whereas this forest has only one connected component in TAP. In our opinion, MAP (like TAP) is an important special case of FAP in the sense that none of the previous approaches (including approaches developed for TAP over two decades) give an approximation guarantee below 2 for MAP.

1.1 Previous literature & possible approaches for MAP

There is extensive literature on approximation algorithms for problems in survivable network design, and also on the minimum-cost 2-ECSS problem including its key special cases (including the unweighted problem, TAP, etc.). To the best of our knowledge, there is no previous publication on FAP or MAP, although the former is well known to the researchers working in this area.

Let us explain briefly why previous approaches do not help for obtaining an approximation guarantee below 2 for MAP. Let G denote the input graph, and let n denote $|V(G)|$. Let opt denote the optimal value, i.e., the minimum cost of a 2-ECSS of the given instance. Recall that the standard cut-covering LP relaxation of the min-cost 2-ECSS problem has a non-negative variable x_e and a cost c_e for each edge e of G, and for each nonempty set of nodes S, $S \neq V$, there is a constraint requiring the sum of the x-values in the cut $(S, V - S)$ to be ≥ 2; the objective is to minimize $\sum_{e \in E} c_e x_e$.

The primal-dual method and the iterative rounding method are powerful and versatile methods for rounding LP relaxations, but in the context of FAP, these methods seem to be limited to proving approximation guarantees of at least 2.

Several intricate combinatorial methods that may also exploit lower-bounds from LP relaxations have been developed for approximation algorithms for unweighted 2-ECSS, e.g., the $\frac{4}{3}$-approximation algorithm of [17]. For unweighted 2-ECSS, there is a key lower bound of n on opt (since any solution must have $\geq n$ edges, each of cost one). This no longer holds for MAP; indeed, the analogous lower bound on opt is $\frac{1}{2}n$ for MAP. It can be seen that an α-approximation

algorithm for unweighted 2-ECSS implies a $(3\alpha - 2)$-approximation algorithm for MAP. (We sketch the reduction: let M denote the set of zero-cost edges in an instance of MAP; observe that $|M| \leq$ opt ; we subdivide (once) each edge in M, then we change all edge costs to one, then we apply the algorithm for unweighted 2-ECSS, and finally we undo the initial transformation; the optimal cost of the unweighted 2-ECSS instance is $\leq$ opt $+2|M|$, hence, the solution of the MAP instance has cost $\leq \alpha($ opt $+2|M|) - 2|M| = \alpha$ opt $+(2\alpha - 2)|M| \leq (3\alpha - 2)$ opt .) Thus the $\frac{4}{3}$-approximation algorithm of [17] for unweighted 2-ECSS gives a 2-approximation algorithm for MAP. (Although preliminary results and claims have been published on achieving approximation guarantees below $\frac{4}{3}$ for unweighted 2-ECSS, there are no refereed publications to date, see [17].)

Over the last two decades, starting with the work of [15], a few methods have been developed to obtain approximation guarantees below 2 for TAP. The recent methods of [1, 4] rely on so-called bundle constraints defined by paths of zero-cost edges. Unfortunately, these methods (including methods that use the bundle constraints) rely on the fact that the set of zero-cost edges forms a connected graph that spans all the nodes, see [4, 12, 15]. Clearly, this property does not hold for MAP.

1.2 Hardness of approximation of MAP and FAP

MAP is a generalization of the unweighted 2-ECSS problem (consider the special case of MAP with $M = \emptyset$). The latter problem is known to be APX-hard; thus, it has a "hardness of approximation" threshold of $1+\epsilon$ where $\epsilon > 0$ is a constant, see [7]. Hence, MAP is APX-hard.

Given the lack of progress on approximation algorithms for FAP, one may wonder whether there is a "hardness of approximation" threshold that would explain the lack of progress. Unfortunately, the results and techniques from the area of "hardness of approximation" are far from the known approximation guarantees for many problems in network design. For example, even for the notorious Asymmetric TSP (ATSP), the best "hardness of approximation" lower bound known is around $\frac{75}{74} \approx 1.014$, see [11].

1.3 Our method for MAP

This paper has the same main result as the joint work with J.Cheriyan, F.Grandoni, A.Khan, and V.V.Narayan [21]. However, the algorithm and analysis are different and there is potential to generalize the methods of this thesis to FAP.

We first present reductions from any given instance of MAP to an instance of MAP without any cut-nodes, unit-splits, zero-splits, redundant-cycles, split-cycles, and multi-edges. Then we present an approximation algorithm with cost $\frac{7}{4}$ opt -2 for instances without these structures, and also prove it can be extend to general MAP instances. Our algorithm starts with a so-called D2 (this is a

min-cost 2-edge cover) and F2, the graph resulting from contracting all blocks of D2. All changes to F2 are also applied to D2. We first alter F2 by covering its bridges until F2 is a set of isolated vertices. We then contract cycles cycles of those vertices to F2 and contract them until F2 is a single vertex. When F2 is a single vertex, D2 is a block containing all vertices of the graph, meaning it is a 2ECSS.

Our presentation is self-contained and formally independent of Vempala & Vetta's manuscript [18]; also, we address a weighted version of the 2-ECSS problem and our challenge is to improve on the approximation guarantee of 2, whereas Vempala & Vetta's goal is to achieve an approximation guarantee of $\frac{4}{3}$ for the unweighted 2-ECSS problem.

For the sake of completeness, we have included the proofs of several basic results (e.g., so-called Facts); these should not be viewed as new contributions.

An outline of the paper follows. Chapter 2 has standard definitions and some preliminary results. Chapter 3 presents an outline of our algorithm for MAP. Chapter 4 presents the preprocessing steps that remove cut-nodes, unit-splits, zero-splits, redundant-cycles, split-cycles, and multi-edges. Chapter 5 creates the subgraph whose cost is a lower bound on all 2ECSS. Chapter 6 alters the subgraph until it is a collection of blocks. Chapter 7 alters the subgraph until it is a 2ECSS.

Chapter 2

Preliminaries

This section has definitions and preliminary results. Our notation and terms are consistent with [2], and readers are referred to that text for further information.

Let $G = (V, E)$ be a (loop-free) multi-graph with edges of cost either zero or one such that the edges of cost zero form a matching. We take G to be the input graph, and we use n to denote $|V(G)|$. Let M denote the set of edges of cost zero. Throughout, the reader should keep in mind that M is a matching; this fact is used in many of our proofs without explicit reminders. We call an edge of M a *zero-edge* and we call an edge of $E - M$ a *unit-edge*. We call a pair of parallel edges a $\{0, 1\}$-edge-pair if one of the two edges of the pair has cost zero and the other one has cost one.

We use the standard notion of contraction of an edge, see [16, p.25]: Given a multi-graph H and an edge $e = vw$, the contraction of e results in the multi-graph $H/(vw)$ obtained from H by deleting e and its parallel copies and identifying the nodes v and w. (Thus every edge of H except for vw and its parallel copies is present in $H/(vw)$; we disallow loops in $H/(vw)$.)

For a graph H and a set of nodes $S \subseteq V(H)$, $\delta_H(S)$ denotes the set of edges that have one end node in S and one end node in $V(H) - S$; moreover, $H - S$ denotes $H[V(H) - S]$, the subgraph of H induced by $V(H) - S$. For a graph H and a set of edges $F \subseteq E(H)$, $H - F$ denotes the graph $(V(H),\ E(H) - F)$. We use relaxed notation for singleton sets, e.g., we use $\delta_H(v)$ instead of $\delta_H(\{v\})$, we use $H - v$ instead of $H - \{v\}$, and we use $H - e$ instead of $H - \{e\}$.

We denote the cost of an edge e of G by c_e. For a set of edges $F \subseteq E(G)$, $c(F) := \sum_{e \in F} c_e$, and for a subgraph G' of G, $c(G') := \sum_{e \in E(G')} c_e$.

For ease of exposition, we often denote an instance G, M by G; then, we do not have explicit notation for the edge costs of the instance, but the edge costs are given implicitly by $c : E(G) \to \{0, 1\}$, and M is given implicitly by $\{e \in E(G) : c_e = 0\}$. Also, we may not distinguish between a subgraph and its node set; for example, given a subgraph H that contains nodes $v_1, v_2, v_3, v_4, \ldots$ we may say that $\{v_1, v_2, v_3\}$ is contained in H.

A *block* is a maximal connected subgraph without a cut-node. Thus, every block of a graph is either a maximal 2-connected subgraph, or a bridge (with

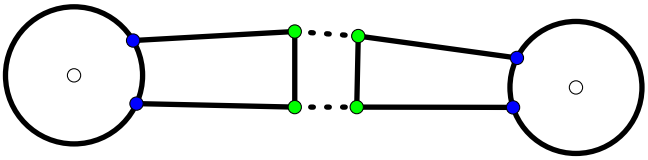


Figure 2.1: In the above figure, unit-edges are drawn as solid lines, zero-edges are drawn as dotted lines, and large circles represent blocks. The green cycle is a Split-cycle. It's contraction results in a cut-node separating two non-trivial biconnected components

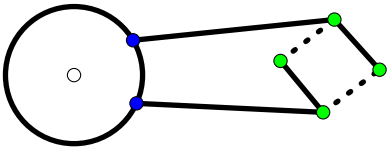


Figure 2.2: In the above figure, unit-edges are drawn as solid lines, zero-edges are drawn as dotted lines, and large circles represent blocks. The green cycle is a Redundant-cycle.

its ends), or an isolated node.

A *biconnected component* (sometimes known as a 2-connected component) is a maximal biconnected subgraph.

We call a biconnected component with opt ≥ 3 is a non-trivial biconnected component.

We call a block with two unit-edges and no adjacent zero-edges a *split-cycle* if contracting it creates a cut-node separating two non-trivial biconnected components, (see Figure 2.1 and Section 4.5).

We call a 4-cycle with two zero-edges a *redundant-cycle* if it has two non-adjacent vertices that have no neighbours outside the cycle (see Figure 2.2 and Section 4.4).

We call a zero-edge a *zero-split* if its contraction creates a cut-node (see Figure 2.3 and Section 4.3).

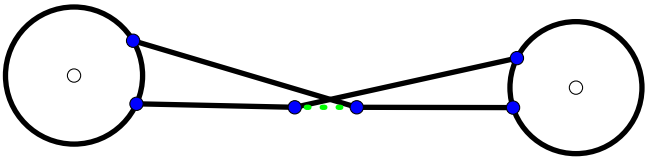


Figure 2.3: In the above figure, unit-edges are drawn as solid lines, zero-edges are drawn as dotted lines, and large circles represent blocks. The green edge is a Zero-split. It's contraction creates a cut-node.

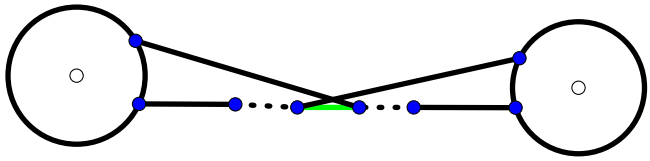


Figure 2.4: In the above figure, unit-edges are drawn as solid lines, zero-edges are drawn as dotted lines, and large circles represent blocks. The green edge is a Unit-split. Its contraction creates a cut-node separating two non-trivial biconnected components, and the adjacent zero-edges are not in the same biconnected component.

We call a unit-edge e a *unit-split* if it has two adjacent zero-edges f_1, f_2 and the contraction of e creates a cut-node separating two non-trivial biconnected components, and f_1 and f_2 are in different biconnected components (see Figure 2.4 and Section 4.2).

2.1 2EC, 2NC, bridges and D2

A multi-graph H is called k-edge connected if $|V(H)| \geq 2$ and for every $F \subseteq E(H)$ of size $< k$, $H - F$ is connected. Thus, H is 2-edge connected if it has ≥ 2 nodes and the deletion of any one edge results in a connected graph. A multi-graph H is called k-node connected if $|V(H)| > k$ and for every $S \subseteq V(H)$ of size $< k$, $H - S$ is connected. We use the abbreviations *2EC* for "2-edge connected," and *2NC* for "2-node connected."

We assume w.l.o.g. that the input G is 2-edge connected. Moreover, we assume w.l.o.g. that there are ≤ 2 copies of each edge (in any multi-graph that we consider); this is justified since an edge-minimal 2-ECSS cannot have three or more copies of any edge (see Proposition 2.1.1 below).

For any instance H, let $\text{opt}(H)$ denote the minimum cost of a 2-ECSS of H. When there is no danger of ambiguity, we use opt rather than $\text{opt}(H)$.

By a *bridge* we mean a cut edge, i.e., an edge of a connected (sub)graph whose removal results in two connected components, and by a *cut node* we mean a node of a connected (sub)graph whose deletion results in two or more connected components. We call a bridge of cost zero a *zero-bridge* and we call a bridge of cost one a *unit-bridge*.

By a *2ec-block* we mean a maximal connected subgraph with two or more nodes that has no bridges. (Observe that each 2ec-block of a graph H corresponds to a connected component of order ≥ 2 of the graph obtained from H by deleting all bridges.) We call a 2ec-block *pendant* if it is incident to exactly one bridge.

The next result characterizes edges that are not essential for 2-edge connectivity.

2.1.1 Proposition. *Let H be a 2EC graph and let $e = vw$ be an edge of H. If $H - e$ has two edge-disjoint v, w paths, then $H - e$ is 2EC.*

By a *2-edge cover* (of G) we mean a set of edges F of G such that each node v is incident to at least two edges of F (i.e., $F \subseteq E(G) : |\delta_F(v)| \geq 2, \forall v \in V(G)$). By $D2(G)$ we mean any minimum-cost 2-edge cover of G (G may have several minimum-cost 2-edge covers, and $D2(G)$ may refer to any one of them); we use $c(D2(G))$ to denote the cost of $D2(G)$; when there is no danger of ambiguity, we use D2 rather than $D2(G)$, and we use $c(D2)$ rather than $c(D2(G))$. Note that D2 may have several connected components, and each may have one or more bridges; moreover, if a connected component of D2 has a bridge, then it has two or more pendant 2ec-blocks.

The next result follows from Theorem 34.15 in [16, Chapter 34].

2.1.2 Proposition. *There is a polynomial-time algorithm for computing D2.*

The next result states the key lower bound used by our approximation algorithm.

2.1.3 Fact. *Let H be any 2EC graph. Then we have $opt(H) \geq c(D2(H))$.*

By a *bridgeless 2-edge cover* (of G) we mean a 2-edge cover (of G) that has no bridges; note that we have no requirements on the cost of a bridgeless 2-edge cover. We mention that the problem of computing a bridgeless 2-edge cover of minimum cost is NP-hard (there is a reduction from TAP), and there is no approximation algorithm known for the case of nonnegative costs.

2.2 Ear decompositions

An *ear decomposition* of a graph is a partition of the edge set into paths or cycles, $P_0, P_1, \ldots, P_k$, such that P_0 is the trivial path with one node, and each P_i ($1 \leq i \leq k$) is either (1) a path that has both end nodes in $V_{i-1} = V(P_0) \cup V(P_1) \cup \ldots \cup V(P_{i-1})$ but has no internal nodes in V_{i-1}, or (2) a cycle that has exactly one node in V_{i-1}. Each of $P_1, \ldots, P_k$ is called an *ear*; note that P_0 is not regarded as an ear. We call $P_i, i \in \{1, \ldots, k\}$, an *open ear* if it is a path, and we call it a *closed ear* if it is a cycle. An *open* ear decomposition $P_0, P_1, \ldots, P_k$ is one such that all the ears $P_2, \ldots, P_k$ are open. (The ear P_1 is always closed.)

2.2.1 Proposition (Whitney [19])**.** *(i) A graph is 2EC iff it has an ear decomposition.*

(ii) A graph is 2NC iff it has an open ear decomposition.

2.3 Polynomial-time computations

All of the computations in this paper can be easily implemented in polynomial time, see [16]. We state this explicitly in all relevant results but we do not elaborate on this elsewhere.

At various points, we need to check if an optimal 2ECSS of a subgraph has cost ≥ 3. This can be done by brute force in polynomial time. If no two unit-edges form a 2ECSS it is the case that opt 3, otherwise it is not the case.

Chapter 3

Outline of the algorithm

We first present reductions from any given instance of MAP to an instance of MAP without any cut-nodes, unit-splits, zero-splits, redundant-cycles, split-cycles, and multi-edges. Then we present an approximation algorithm with cost $\frac{7}{4}$ opt -2 for instances without these structures, and also prove it can be extend to general MAP instances. Our algorithm starts with a so-called D2 (this is a min-cost 2-edge cover) and F2, the graph resulting from contracting all blocks of D2. All changes to F2 are also applied to D2. We first alter F2 by covering its bridges until F2 is a set of isolated vertices. We then contract cycles cycles of those vertices to F2 and contract them until F2 is a single vertex. When F2 is a single vertex, D2 is a block containing all vertices of the graph, meaning it is a 2ECSS.

Algorithm (outline):

1. Apply Preprocessing, reducing the MAP instance to a set of MAP instances without cut-nodes, unit-splits, zero-splits, redundant-cycles, split-cycles and multi-edges.
2. Perform Base Graph Construction, which computes D2 and F2, lower bounds on the optimal solution, and introduces "credit", which can be traded to "buy" more edges.
3. Perform Bridge-Covering, which iteratively trades credit to add paths of edges to D2 and F2. This repeats until D2 and F2 no longer contain any bridges.
4. Perform Gluing, which iteratively trades credit to add cycles of edges to D2 and F2. This repeats until D2 and F2 are connected with no bridges.
5. Output the 2ECSS for the MAP instance obtained from undoing the reductions for multi-edges, split-cycles, redundant-cycles, zero-splits, unit-splits, and cut-nodes.

3.1 Preprocessing

The goal of this subroutine is to break the given MAP instance into manageable subproblems. We aim to find a subgraph in polytime which lower bounds a 2ECSS. Were we to do this on an arbitrary MAP instance, the edges required to augment it to a 2ECSS might cost $\geq opt$, meaning we would not beat a factor 2 approximation. When we can guarantee that a MAP instance has none of the elements we preprocess, our algorithm augments a base graph to a 2ECSS using at most $\frac{3}{4}opt$ edges.

This subroutine has 6 stages. If we are given a MAP instance that we cannot solve optimally, i.e. more than a constant number of vertices, then we perform six different reductions. Each of the six stages performs one of the reductions until it is no longer possible.

Stage 1 handles cut-nodes. To handle them, we treat each biconnected component of the graph as its own graph. We solve each recursively and then combine the solutions into a solution for the original graph.

Stage 2 handles unit-splits. To handle them, we contract them, and deal with the resulting cut-node as in stage 1.

Stage 3 handles zero-splits.To handle them, we contract them, and deal with the resulting cut-node as in stage 1.

Stage 4 handles redundant-cycles. To handle them, we contract them

Stage 5 handles split-cycles. To handle them, we contract them, and deal with the resulting cut-node as in stage 1.

Stage 6 handles unit-multiedges. To handle them, we delete them. We solve the resulting graph, which is also a solution to the original graph.

In Chapter 4 we elaborate on these stages, one in each section. A MAP instance with the first i elements eliminated is called a MAPi instance. We prove that we can eliminate all these elements in polytime. Moreover, we prove that each of these stages preserves our approximation guarantee. That is, if a MAP$i+1$ can be approximated within factor $\frac{7}{4}$, then so can a MAPi instance.

3.2 Base Graph Construction

In this subroutine, the goal is to create a subgraph which will become our solution. It will have the same cost as our solution, thanks to the concept of *credit*. Credit is a value placed on the vertices of a graph that contributes to the weight of the graph. Its purpose is to be exchanged for edges, so we can add edges to a graph without increasing its weight. This subroutine creates two graphs. D2, which will become a 2ECSS, and F2, which is just D2 but with the blocks contracted. We introduce F2 to more easily describe the algorithm and maintain *credit invariants*, which are introduced in stage 5 below.

This subroutine has 5 stages. If we are given a preprocessed MAP instance, then we create two graphs D2, and F2.

The first graph, D2, requires three stages to create.

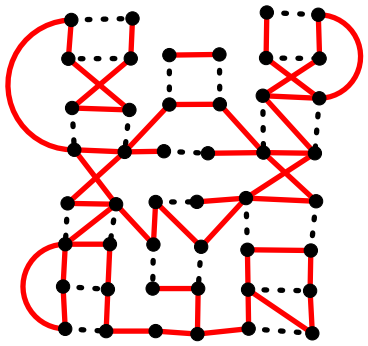


Figure 3.1: The above figure illustrates an Example Graph. Unit-edges are drawn as solid lines, and zero-edges are drawn as dotted lines. The blue and black edges are part of D2(and F2), while the red unit-edges are not. This graph does not require any preprocessing

Stage 1 finds a minimal 2-edge cover. This graph's cost is a lower bound on the cost of an optimal 2ECSS because all vertices in a 2ECSS have degree at-least 2.

Stage 2 handles bad-triangles. To handle them, we swap one of the unit-edges of the triangle for a different unit-edge.

Stage 3 adds credit. Each vertex receives $\frac{3}{8}$ credit for every unit edge of the 2-edge cover adjacent to it. This makes the graph plus the credit a lower bound on $\frac{7}{4}$ times the cost of an optimal 2ECSS.

The second graph, F2, requires two stages to create.

Stage 4 contracts the blocks of D2.

Stage 5 establishes credit invariants: all trees have 1 credit, all special vertices have .5 credit, all other contracted vertices have 1 credit, all uncontracted vertices have $\frac{3}{8}$ credit for each adjacent unit edge.

Finally, in what follows we refer to graph that results from the input graph by contracting all blocks of D2. We call this graph G'.

In Figure 3.2, the first subfigure contains a minimal 2-edge cover, marked by blue unit-edges and dotted zero-edges. In the second subfigure, a red edge and blue edge are swapped on the right side of the figure. This corresponds to the only bad-triangle in the 2-edge cover, which we handle with this swap. The third figure we add credit. $\frac{3}{8}$ per adjacent unit edge for each vertex, though we omit the denominator of 8 for a clearer picture. At this point, we have D2. In the fourth subgraph, we contract all blocks in D2. In the fifth, we reorganize credit, giving 1 to each tree. This completes the Base Graph Stage, as we have created F2.

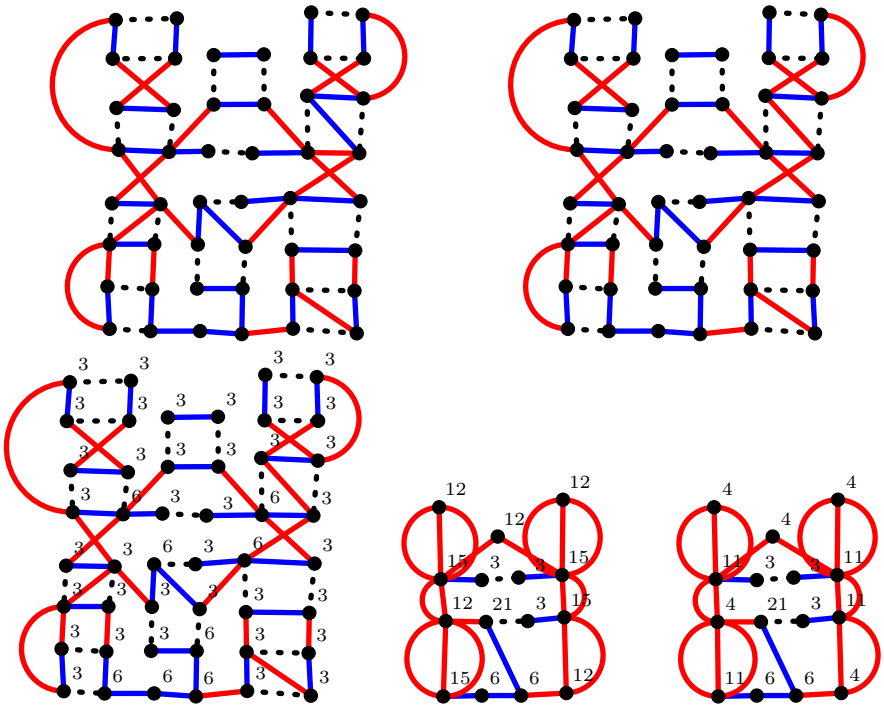


Figure 3.2: The above figure illustrates the Base Graph Construction Stage on the Example Graph. The figures are read from left to right, top to bottom. Unit-edges are drawn as solid lines, and zero-edges are drawn as dotted lines. The blue and black edges are part of D2(and F2), while the red unit-edges are not. The numbers represent the amount of $\frac{1}{8}$ credits on each vertex. For details, see the discussion in the text of Section 3.2

3.3 Bridge Covering

In this subroutine, the goal is to alter F2 and D2 until they contain no more bridges. We work with F2 in G' because the credit invariants are useful for our algorithm. But all edges added to F2 and all credit removed from F2 are also added and removed from D2 in G. The key to altering bridges is to add paths to F2 that start at a leaf l of a tree T in F2 and end at some vertex u in T. Thus all the edges of T between l and u are no longer bridges. We select u so that credit of all vertices between u and l help pay for the path we add, thereby keeping all credit invariants intact.

To cover all bridges from F2, select a tree T of F2 such that T has bridges. Select a leaf l in T. For each vertex v in T, let P_v be the path from v to l in T.

Next we need to select a vertex $u \in T$ such that:

1. There is a path from u to l in $G' - E(F2)$

2. P_u satisfies one of the following:

   (a) P_u contains ≥ 2 contracted vertices

   (b) P_u has ≥ 3 edges

   (c) P_u has exactly 2 edges and P_u contains a vertex, (possibly one of its end vertices) which has ≥ 2 adjacent unit edges in T.

Once we have u, we choose a path from u to l internally disjoint from T that uses the fewest edges of G'. We'll call this path Q_u. We add Q_u to F2, contract the resulting cycle created by Q_u and P_u. Finally we remove $|Q_u|$ credit from the resulting contracted vertex. This process covers all the bridges in P_u and does not add any.

We prove in Chapter 6 that we can always do this in polytime as long as F2 has bridges, and that doing so maintains the credit invariants.

In Figure 3.3, the first subfigure we continue the algorithm by adding a path to F2 to create a cycle. The path of bridges P_{u_1} that we cover satisfies $|P_{u_1}| \geq 3$ and P_{u_1} has ≥ 2 contracted vertices. We contract the cycle we've created in the second subfigure. In the third subfigure, we cover another path P_{u_2} satisfying P_{u_2} has ≥ 2 contracted vertices, and $|P_{u_2}| = 2$ and it contains a vertex which has ≥ 2 adjacent unit edges in T. Again we contract in the fourth subfigure. At this point, we've covered all bridges in one tree. We move onto the final tree, where we cover a final path P_{u_3} such that $|P_{u_3}| \geq 3$ and P_{u_3} has ≥ 2 contracted vertices. The final subfigure contracts the resulting cycle and marks the end of the Bridge Covering Stage, as there are no more bridges.

3.4 Gluing

In this subroutine, the goal is to continue altering F2 until it is a single vertex. To do so, we first contract cycles in G' of length > 3 until none remain. After that, we can contract cycles of length 3 in G' until none remain. We show in Chapter 7 that doing so maintains credit invariants in F2. This gives us a graph where the only simple cycles are of length 2.

The last step of the gluing stage involves handling all cycles of length 2. The details are covered in Chapter 7. Handling all cycles of length 2 means that G' and thus F2 will be a single vertex. This means D2 is a block containing all vertices of G, which is a 2ECSS. This completes the algorithm.

In Figure 3.4, our graph G' has only cycles of length 2. Therefore, in the first subfigure, we find a cycle where both vertices are non-special contracted vertices and add it. We then contract it in the second subfigure. At, this point, there are no more cycles of non-special contracted vertices. Thus in the third subfigure, we uncontract a special vertex, resulting in a 4-cycle. In subfigure four, we carefully pick 2 red edges to "buy" in exchange for 1 blue edge and 1 credit, in order to 2-edge connect the 4-cycle to the non-special vertex. Finally, we contract this new block in subfigure five. Subfigures 6-8 are

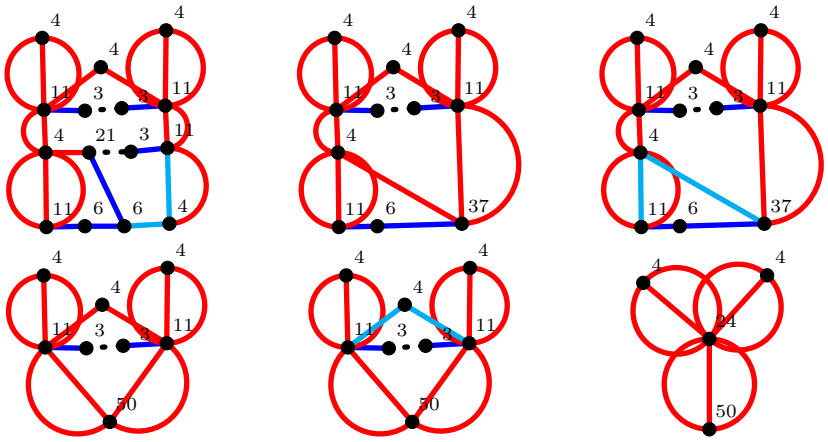


Figure 3.3: The above figure illustrates the Bridge Covering Stage on the Example Graph. The figures are read from left to right, top to bottom. Unit-edges are drawn as solid lines, and zero-edges are drawn as dotted lines. The blue and black edges are part of D2(and F2), and light blue means the edges were just added, while the red unit-edges are not in D2(or F2). The numbers represent the amount of $\frac{1}{8}$ credits on each vertex. For details, see the discussion in the text of Section 3.3

identical to Subfigures 3-5. However, subfigure 8 completes the Gluing Stage of the algorithm, because F2 is a single vertex at that point.

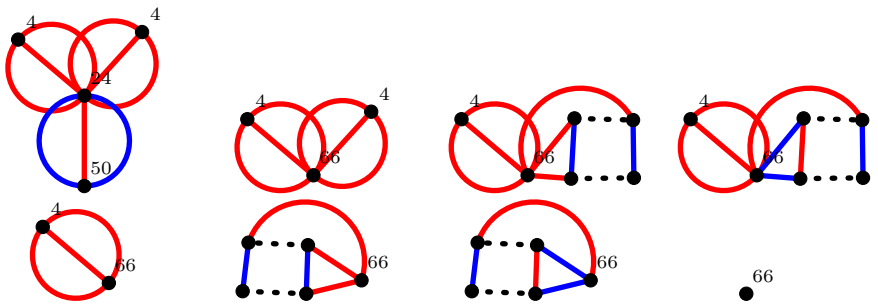


Figure 3.4: The above figure illustrates the Gluing Stage on the Example Graph. The figures are read from left to right, top to bottom. Unit-edges are drawn as solid lines, and zero-edges are drawn as dotted lines. The blue and black edges are part of D2(and F2), while the red unit-edges are not. The numbers represent the amount of $\frac{1}{8}$ credits on each vertex. For details, see the discussion in the text of Section 3.4

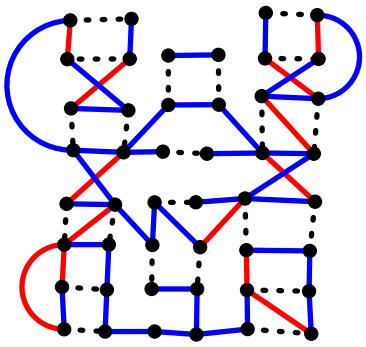


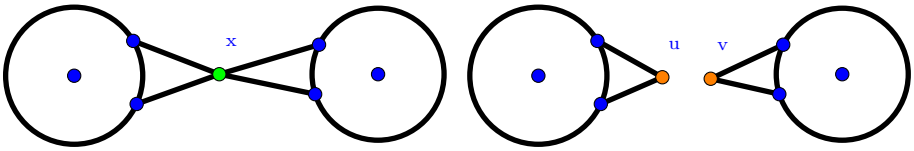
Figure 3.5: The above figure illustrates a 2ECSS found algorithmically on the Example Graph. Unit-edges are drawn as solid lines, and zero-edges are drawn as dotted lines. The blue and black edges are part of D2(and F2), while the red unit-edges are not.

Chapter 4

Pre-processing

In this section, we describe several reductions which we can apply to a MAP instance. The algorithm will remove all instances of the first reduction, then all instances of the second reduction and so on. We only apply reductions to MAP instances with 20 or more vertices, as all others can be solved optimally in constant time. There is one set of terms we define here, and that is a MAP instance where the first i reductions cannot be applied are also MAPi instances for $1 \leq i \leq 6$. All remaining definitions and proofs relating to each of these reductions are contained in the respective subsections. For the more complicated reductions, we include figures showing what a graph looks like before and after that reduction is applied.

4.1 Handling cut-nodes



(a) Find a cut-node (green) (b) Separate biconnected components

Figure 4.1: The above figure illustrates Handling Cut-Nodes. Unit-edges are drawn as solid lines, zero-edges are drawn as dotted lines, and large circles represent blocks. The green vertex x is a cut-node. We create duplicates u, v (orange) of x in order to separate all maximal biconnected components surrounding x.

4.1.1 Lemma. *We can find and remove a cut-node, or prove none exist, in polytime*

Proof. There are n potential cut-nodes. We can check to see if deleting a vertex disconnects the graph in polytime. If it does, we can decompose the graph G by treating the biconnected components as individual graphs G_1,..., G_k. □

4.1.2 Lemma. *Removing a cut-node preserves the matching property of zero-edges.*

Proof. Decomposing the graph does not make any zero-edges adjacent to each other. □

4.1.3 Lemma. *Given 2ECSS's S_1, ..., S_k for G_1,..., G_k with costs $s(G_1)$, ...,$s(G_k)$ respectively, we can find a 2ECSS S for G with cost $\sum_{i=1}^{k} s(G_i)$*

Proof. Let $S = \bigcup_{i=1}^{k} S_1$. Then S is a 2ECSS for G because if S can be disconnected by deleting < 2 edges, then so can some $S_i, 1 \leq i \leq k$, which is a contradiction. Clearly, S has cost $\sum_{i=1}^{k} s(G_i)$, as desired. □

4.1.4 Lemma. $\sum_{i=1}^{k}$ *opt* $(G_i) \leq$ *opt* (G)

Proof. Given an optimal solution S for G, all the cut-nodes of G must be cut-nodes of S, therefore decomposing the solution on these cut-nodes gives us a solution S_i for each of the biconnected component G_i with cost $s(G_i)$. We have opt $(G_i) \leq s(G_i)$ because of optimality, therefore $\sum_{i=1}^{k}$ opt $(G_i) \leq \sum_{i=1}^{k} s(G_i) =$ opt (G) □

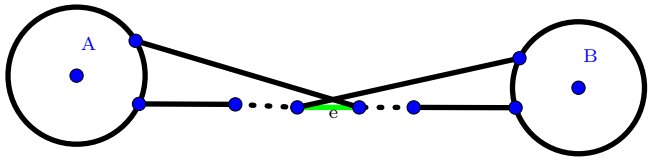
4.1.5 Lemma. *If we have a polytime algorithm for MAP1 with cost $\frac{7}{4}$ opt $(G)-2$, then we have a polytime algorithm for MAP with cost $\frac{7}{4}$ opt $(G) - 2$*

Proof. Proof by induction. Let G be a minimal counter example. G must have a cut-node, which we can remove in polytime. Removing the cut-node gives us graphs G_1,...,G_k. We can find solutions S_1,..,S_k in polytime by induction. Either we solve G_i optimally for all i, in which case our solution S for G has cost opt $(G) \leq \frac{7}{4}$ opt $(G) - 2$, or without loss of generality, our solution S_1 for G_1 has cost $\frac{7}{4}$ opt $(G_1) - 2$. All other components are solved either optimally or algorithmically, and therefore will have cost $\leq \frac{7}{4}$ opt . Thus by our above result, we can find a 2ECSS for G in polytime with cost $\leq \sum_{i=1}^{k} \frac{7}{4}$ opt $(G_i) - 2 \leq \frac{7}{4}$ opt $(G) - 2$, which completes the proof. □

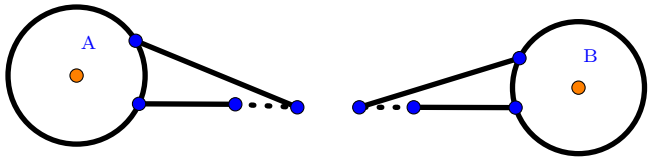
4.2 Handling unit-splits

A unit-split is a unit-edge whose contraction creates a cut-node with two adjacent zero-edges, such that two of the resulting biconnected components have opt ≥ 3, and the zero-edges are in different biconnected components.

4.2.1 Lemma. *We can find and remove a unit-split, or prove there are none, in polytime*



(a) Find a unit-split e (green)



(b) Contract and separate biconnected components A,B (orange)

Figure 4.2: The above figure illustrates Handling Unit-Splits. Unit-edges are drawn as solid lines, zero-edges are drawn as dotted lines, and large circles represent blocks. The green edge e is a Unit-Split. Contracting it creates a cut-node, which we handle identically to any other cut node.

Proof. There are m potential unit-splits. We can contract each in constant time and check if doing so creates a cut-node. If it does, we can also check if the two adjacent zero-edges are in different biconnected components in constant time, and finally we can check if some G_1 and G_2 have opt (G_1), opt $(G_2) \geq 3$. If all of these conditions are satisfied, we can decompose the graph G by treating the biconnected components as individual graphs G_1, G_2,..., G_k. Otherwise we uncontract and move to the next potential unit-split. □

4.2.2 Lemma. *Removing a unit-split preserves the matching property of zero-edges, and creates no cut-nodes*

Proof. Our contraction only makes two zero-edges adjacent, but when we decompose the graph, those zero-edges are no longer adjacent, therefore the matching property is restored. There are no cut-nodes in the original graph, therefore the only potential cut-node is the newly created node, but because of our decomposition, we know it is not a cut-node. Therefore no cut-nodes are created. □

4.2.3 Lemma. *Given 2ECSS's $S_1, ..., S_k$ for G_1,..., G_k with costs $s(G_1)$, ..., $s(G_k)$ respectively, we can find a 2ECSS S for G with cost $\sum_{i=1}^{k} s(G_i) + 2$*

Proof. Let e be the contracted unit-edge. Let $S^* = \bigcup_{i=1}^{k} S_1$. We showed in a previous section that S^* is a 2ECSS for G/e. Then $S^* + e$ has at most 1 bridge in G which is e, as otherwise S^* would have a bridge in G/e. Let f be any edge between the two blocks of $S^* + e$. Let $S = S^* + e + f$. We showed before that S^* has cost $\sum_{i=1}^{k} s(G_i)$, therefore S has cost $\sum_{i=1}^{k} s(G_i) + 2$, as desired. □

4.2.4 Lemma. $\sum_{i=1}^{k}$ *opt* $(G_i) \leq$ *opt* (G)

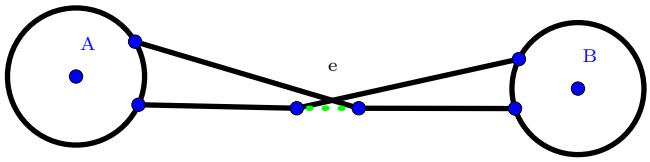
Proof. Given an optimal solution S for G, we can contract a unit-split e, creating a cut-node in G and S. We showed in a previous section that $\sum_{i=1}^{k}$ opt $(G_i) \leq$ opt (G/e). Therefore $\sum_{i=1}^{k}$ opt $(G_i) \leq$ opt (G). □

4.2.5 Lemma. *If we have a polytime algorithm for MAP2 with cost* $\frac{7}{4}$ *opt* -2*, then we have a polytime algorithm for MAP1 with cost* $\frac{7}{4}$ *opt* -2

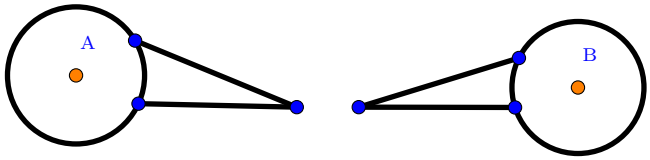
Proof. Proof by induction, let G be a minimal counter example. G must have a unit-split, which we can remove in polytime. Removing the split gives us graphs G_1,...,G_k. We can find solutions S_1,..,S_k in polytime by induction. Without loss of generality, S_1 will have cost $\frac{7}{4}$ opt $(G_1) - 2$ and S_2 will have cost $\frac{7}{4}$ opt $(G_2) - 2$. All other components are solved either optimally or algorithmically, and therefore will have cost $\leq \frac{7}{4}$ opt . Thus by our above result, we can find a 2ECSS for G in polytime with cost $\leq \sum_{i=1}^{k} \frac{7}{4}$ opt $(G_i) - 4 + 2 \leq \frac{7}{4}$ opt $(G) - 2$, which completes the proof. □

4.3 Handling zero-splits

A zero-split is a zero-edge whose contraction creates a cut-node



(a) Find a zero-split e (green)



(b) Contract and separate biconnected components A,B (orange)

Figure 4.3: The above figure illustrates Handling Zero-Splits. Unit-edges are drawn as solid lines, zero-edges are drawn as dotted lines, and large circles represent blocks. The green edge e is a Zero-Split. Contracting it creates a cut-node, which we handle identically to any other cut node.

4.3.1 Lemma. *We can find and remove a zero-split in polytime*

Proof. There are $\frac{n}{2}$ potential zero-splits. We can contract each in constant time and check if doing so creates a cut-node. If all of these conditions are satisfied, we leave the zero-split contracted and decompose the graph G by treating the biconnected components as individual graphs G_1, G_2,..., G_k. Otherwise we uncontract and move to the next potential zero-split. □

4.3.2 Lemma. *Removing a zero-split preserves the matching property of zero-edges and does not create unit-splits or cut-nodes*

Proof. Our contraction does not make any zero-edges adjacent therefore the matching is maintained. There are no cut-nodes in the original graph, therefore the only potential cut-node is the newly created node, but because of our decomposition, we know it is not a cut node. Therefore no cut-nodes are created. There are no unit-splits in the original graph, and unit-splits only involve vertices with an adjacent zero-edge. No such vertices are created, therefore no unit-splits are created. □

4.3.3 Lemma. *Given 2ECSS's $S_1, ..., S_k$ for G_1,..., G_k with costs $s(G_1)$,..., $s(G_k)$ respectively, we can find a 2ECSS S for G with cost $\sum_{i=1}^{k} s(G_i) + 1$*

Proof. Let e be the contracted zero-edge. Let $S^* = \bigcup_{i=1}^{k} S_1$. We showed in the previous section that S^* is a 2ECSS for G/e. Then $S^* + e$ has at most 1 bridge in G which is e, as otherwise S^* would have a bridge in G/e. Let f be any edge between the two blocks of $S^* + e$. Let $S = S^* + e + f$. We showed before that S^* has cost $\sum_{i=1}^{k} s(G_i)$, therefore S has cost $\sum_{i=1}^{k} s(G_i) + 1$, as desired. □

4.3.4 Lemma. $\sum_{i=1}^{k} \ opt\ (G_i) \leq \ opt\ (G)$

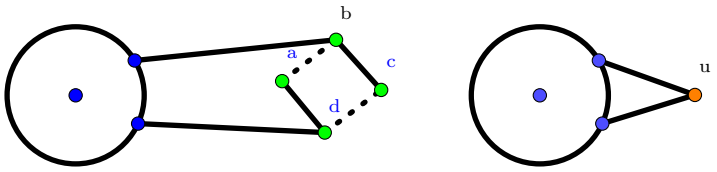
Proof. Given an optimal solution S for G, we can contract a zero-split, creating a cut-node in G and S. Decomposing S on this cut-node gives us a solution S_i for each of the biconnected component G_i with cost $s(G_i)$. We have $\text{opt}\ (G_i) \leq s(G_i)$ because of optimality, therefore $\sum_{i=1}^{k} \text{opt}\ (G_i) \leq \sum_{i=1}^{k} s(G_i) = \text{opt}\ (G)$ □

4.3.5 Lemma. *If we have a polytime algorithm for MAP3 with cost $\frac{7}{4}\ opt\ - 2$, then we have a polytime algorithm for MAP2 with cost $\frac{7}{4}\ opt\ - 2$*

Proof. Proof by induction, let G be a minimal counter example. G must have a unit-split, which we can remove in polytime. Removing the split gives us graphs G_1,...,G_k. We can find solutions S_1,..,S_k in polytime by induction. Either we solve G_i optimally for all i, in which case our solution S for G has cost $\text{opt}\ (G) + 1 \leq \frac{7}{4}\ \text{opt}\ (G) - 2$, or without loss of generality our solution S_1 for G_1 has cost $\frac{7}{4}\ \text{opt}\ (G_1) - 2$. All other components are solved either optimally or algorithmically, and they all have a vertex with no adjacent zero-edges, therefore they all have cost $\leq \frac{7}{4}\ \text{opt}\ - 1$. Thus by our above result, we can find a 2ECSS for G in polytime with cost $\leq \sum_{i=1}^{k} \frac{7}{4}\ \text{opt}\ (G_i) - 3 + 1 \leq \frac{7}{4}\ \text{opt}\ (G) - 2$, which completes the proof. □

4.4 Handling redundant-cycles

A redundant-cycle is a 4-cycle with two zero-edges with two vertices that have no neighbours outside the cycle and are not adjacent.



(a) Find a redundant cycle abcd (green) (b) Contract abcd into vertex u (orange)

Figure 4.4: The above figure illustrates Handling Redundant-Cycles. Unit-edges are drawn as solid lines, zero-edges are drawn as dotted lines, and large circles represent blocks. The green cycle abcd is a redundant-cycle. Vertices a and c are non-adjacent and have no neighbours outside abcd. We contract abcd into vertex u. If u is a cut-vertex, we handle it as any other.

4.4.1 Lemma. *We can find and remove a redundant-cycle, or prove one does not exist, in polytime*

Proof. There are n^2 potential redundant-cycles. We can check 2 zero-edges to see if they form a cycle where two vertices have neighbourhoods which meet the requirements in constant time. If all of the conditions are satisfied, we leave the redundant-cycle contracted. We label the biconnected components in the resulting graph $G_1, ..., G_k$, and we decompose the graph if $k > 1$. Otherwise we uncontract and move to the next potential redundant-cycle. □

4.4.2 Lemma. *Removing a redundant-cycle preserves the matching property of zero-edges and does not create unit-splits, zero-splits,or cut-nodes*

Proof. Our contraction does not make any zero-edges adjacent and creates a vertex with no adjacent zero-edges, therefore the matching is maintained. There are no cut-nodes in the original graph, therefore the only potential cut-node is the newly created node, but because of our decomposition, we know it is not a cut node. Therefore no cut-nodes are created. There are no unit-splits or zero-splits in the original graph, and both only involve vertices with an adjacent zero-edge. No such vertices are created, therefore no unit-splits or zero-splits are created. □

4.4.3 Lemma. *Given 2ECSS's $S_1, ..., S_k$ for G_1,...,G_k with costs $s(G_1)$,...,$s(G_k)$ respectively, we can find a 2ECSS S for G with cost $\sum_{i=1}^{k} s(G_i) + 2$*

Proof. Let C be the contracted cycle. Let $S^* = \bigcup_{i=1}^{k} S_1$. We showed in a previous section that S^* is a 2ECSS for G/C. Then $S^* + C$ has no bridges, as otherwise S^* would have a bridge in G/C. We showed before that S^* has cost $\sum_{i=1}^{k} s(G_i)$, therefore S has cost $\sum_{i=1}^{k} s(G_i) + 2$, as desired. □

4.4.4 Lemma. $\sum_{i=1}^{k} \ opt\ (G_i) + 2 \leq \ opt\ (G)$

Proof. Given an optimal solution S for G, we can contract a redundant-cycle C in G and decompose if a cut-node is created. We will have contracted two

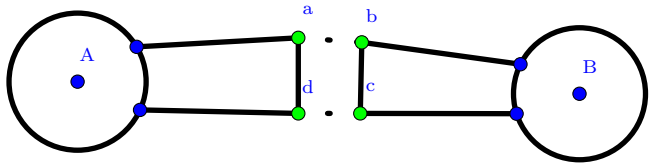
unit-edges of S, because all vertices have degree 2 in S and there are the unadjacent vertices in C with no neighbours outside C. Therefore $\text{opt}\ (G/C)+2 = \text{opt}\ (G)$. We know that $\sum_{i=1}^{k} \text{opt}\ (G_i) \leq \text{opt}\ (G/C)$ from a previous section, therefore $\sum_{i=1}^{k} \text{opt}\ (G_i)+2 \leq \text{opt}\ (G)$ as desired. □

4.4.5 Lemma. *If we have a polytime algorithm for MAP4 with cost $\frac{7}{4}$ opt -2, then we have a polytime algorithm for MAP3 with cost $\frac{7}{4}$ opt -2*

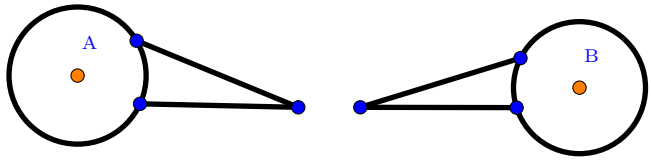
Proof. Proof by induction, let G be a minimal counter example. G must have a redundant-cycle, which we can remove in polytime. Removing the cycle gives us graphs G_1,...,G_k. We can find solutions S_1,..,S_k in polytime by induction. Either we solve G_i optimally for all i, in which case our solution S for G has cost $\leq \sum_{i=1}^{k} \text{opt}\ (G_i)+2 \leq \text{opt}\ (G) \leq \frac{7}{4}\ \text{opt}\ (G)-2$, or without loss of generality our solution S_1 for G_1 has cost $\frac{7}{4}\ \text{opt}\ (G_1)-2$. All other components are solved either optimally or algorithmically, therefore they all have cost $\leq \frac{7}{4}$ opt . Thus by our above result, we can find a 2ECSS for G in polytime with cost $\leq \sum_{i=1}^{k} \frac{7}{4}\ \text{opt}\ (G_i)-2+2 \leq \frac{7}{4}\ \text{opt}\ (G)-2$, which completes the proof. □

4.5 Handling split-cycles

A split-cycle is a block with two unit-edges and no adjacent zero-edges, whose contraction creates a cut-node such that two of the biconnected components have $\text{opt}\ \geq 3$.



(a) Find a split-cycle abcd (green)



(b) Contract and separate biconnected components A,B (orange)

Figure 4.5: The above figure illustrates Handling Split-Cycles. Unit-edges are drawn as solid lines, zero-edges are drawn as dotted lines, and large circles represent blocks. The green cycle *abcd* is a Split-Cycle. Contracting it creates a cut-node, which we handle identically to any other cut node.

4.5.1 Lemma. *We can find and remove a split-cycle, or prove one does not exist, in polytime*

Proof. There are n^2 potential split-cycles. We can contract each in constant time and check if doing so creates a cut-node. If it does, we can check if two of the biconnected components have opt ≥ 3 in polytime. If all of these conditions are satisfied, we leave the split-cycle contracted. We label the biconnected components in the resulting graph $G_1, ..., G_k$ and decompose the graph G into these components. Otherwise we uncontract and move to the next potential split-cycle. □

4.5.2 Lemma. *Removing a split-cycle preserves the matching property of zero-edges and does not create cut-nodes, redundant-cycles, unit-splits or zero-splits*

Proof. Our contraction does not make any zero-edges adjacent therefore the matching is maintained. There are no cut-nodes in the original graph, therefore the only potential cut-node is the newly created node, but because of our decomposition, we know it is not a cut node. Therefore no cut-nodes are created. There are no redundant-cycles, unit-splits or zero-splits in the original graph, and all three only involve vertices with an adjacent zero-edge. No such vertices are created, therefore no unit-splits, zero-splits or redundant-cycles are created. □

4.5.3 Lemma. *Given 2ECSS's $S_1, ..., S_k$ for G_1,..., G_k with costs $s(G_1)$,...,$s(G_k)$ respectively, we can find a 2ECSS S for G with cost $\sum_{i=1}^{k} s(G_i) + 2$*

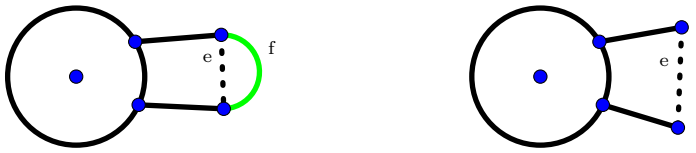
Proof. Let C be the contracted block. Let $S^* = \bigcup_{i=1}^{k} S_1$. We showed in a previous section that S^* is a 2ECSS for G/C. Then $S^* + C$ has no bridges, as otherwise S^* would have a bridge in G/C. We showed before that S^* has cost $\sum_{i=1}^{k} s(G_i)$, therefore S has cost $\sum_{i=1}^{k} s(G_i) + 2$, as desired. □

4.5.4 Lemma. $\sum_{i=1}^{k}$ *opt* $(G_i) \leq$ *opt* (G)

Proof. Given an optimal solution S for G, we can contract a split-cycle C, creating a cut-node. We know that $\sum_{i=1}^{k}$ opt $(G_i) \leq$ opt (G/C) from a previous section. Also, opt $(G/C) \leq$ opt (G), therefore $\sum_{i=1}^{k}$ opt $(G_i) \leq$ opt (G) □

4.5.5 Lemma. *If we have a polytime algorithm for MAP5 with cost $\frac{7}{4}$ opt -2, then we have a polytime algorithm for MAP4 with cost $\frac{7}{4}$ opt -2*

Proof. Proof by induction, let G be a minimal counter example. G must have a split-cycle, which we can remove in polytime. Removing the split gives us graphs G_1,...,G_k. We can find solutions S_1,..,S_k in polytime by induction. Without loss of generality, S_1 will have cost $\frac{7}{4}$ opt $(G_1) - 2$ and S_2 will have cost $\frac{7}{4}$ opt $(G_2) - 2$. All other components are solved either optimally or algorithmically, and therefore will have cost $\leq \frac{7}{4}$ opt . Thus by our above result, we can find a 2ECSS for G in polytime with cost $\leq \sum_{i=1}^{k} \frac{7}{4}$ opt $(G_i) - 4 + 2 \leq \frac{7}{4}$ opt $(G) - 2$, which completes the proof. □



(a) Find a unit multi-edge f (green) (b) Delete the edge f

Figure 4.6: The above figure illustrates Handling Unit Multi-edges. Unit-edges are drawn as solid lines, zero-edges are drawn as dotted lines, and large circles represent blocks. The green edge f is a unit multi-edge. We delete f.

4.6 Handling unit multi-edges

4.6.1 Lemma. *We can find and remove a unit multi-edge, or prove there are none, in polytime*

Proof. There are m potential multi-edges. We can check any edge to see if it is a unit multi-edge in polytime. If it is, we delete it resulting in the graph G_1. □

4.6.2 Lemma. *Removing a unit multi-edge preserves the matching property of zero-edges and does not create unit-splits, zero-splits, split-cycles, cut-nodes, or redundant-cycles*

Proof. Deleting a multi-edge does not alter 2-connectivity or adjacency at all, therefore it cannot destroy the matching property or create any of unit-splits, zero-splits, split-cycles, cut-nodes, or redundant-cycles. □

4.6.3 Lemma. *Given 2ECSS's S_1 for G_1 with costs $s(G_1)$, we can find a 2ECSS S for G with cost $s(G_1)$*

Proof. Let $S = S_1$. S is 2ECSS because S_1 is a 2ECSS, and clearly S has cost $s(G_1)$. □

4.6.4 Lemma. $opt\ (G_1) \leq\ opt\ (G)$

Proof. Let S be an optimal solution for G. Let e, f be multi-edges such that e is a unit-edge and $G - e = G_1$. If S does not contain e, then let $S_1 = S$. If S does not contain f, then $S_1 = S - e + f$. If S_1 contains both, then the only possible bridge in $S - e$ is f, and because G is 2-connected, there is an edge h between the two blocks of $S - e$, thus let $S_1 = S - e + h$. In all cases, S_1 is a 2ECSS for G with cost $\ opt\ (G)$, completing the proof. □

4.6.5 Lemma. *If we have a polytime algorithm for MAP6 with cost $\frac{7}{4}\ opt\ - 2$, then we have a polytime algorithm for MAP5 with cost $\frac{7}{4}\ opt\ - 2$*

Proof. Proof by induction, let G be a minimal counter example. G must have a unit multi-edge, which we can remove in polytime. Removing the split gives us graph G_1. We can find solution S_1 in polytime by induction. S_1 will have cost $\leq \frac{7}{4}$ opt $(G_1) - 2$. Therefore we can find a 2ECSS for G with cost $\leq \frac{7}{4}$ opt $(G) - 2$, which completes the proof. □

Chapter 5

Base Graph Construction

For this section, we need the following definitions: Bad triangles are 3-cycles which contain a zero-edge and a zero-edge as their only adjacent edge. A contracted vertex in some graph is the result of contracting a block of G. An uncontracted vertex in some graph is a vertex of G. A special vertex in some graph is a neighbourless vertex which results from contracting a block with 2 unit-edges in G.

At the start of this section, we have a MAP6 instance, i.e. a MAP instance wherein no reductions are possible. This means that we have at least 20 vertices and thus $\text{opt}\ (G) \geq 10 \implies \frac{7}{4}\ \text{opt}\ (G) - 2 \geq\ \text{opt}\ (G)$.

At the end of this section, we'll have created two graphs. The first is D2, which is a spanning subgraph of G with cost $\leq \frac{7}{4}$ opt (G) in which all vertices have degree ≥ 2 and there are no bad-triangles. Both graphs will have "credit" on them, which increases their cost without increasing their number of edges. The point of the credit is we can remove 1 credit to add 1 unit-edge without increasing the cost of the graph.

The second graph is F2. F2 is a minor of G created by contracting the blocks of D2 (which makes it a forest) and assigning its credit as follows: All trees of F2 have 1 credit, all special vertices of F2 have .5 credit, all other contracted vertices have 1 credit, all uncontracted vertices have $\frac{3}{8}$ credit for each adjacent unit edge. Note that any vertex of F2 with degree < 2 must be a contracted vertex.

We work with F2 because it is easier to manage, but D2 will ultimately be our 2ECSS for the MAP6 instance.

5.0.1 Lemma. *If have a polytime algorithm to find an F2 which is a single vertex, then we have a polytime algorithm for MAP6 with cost* $\frac{7}{4}$ *opt* $(G) - 2$.

Proof. If F2 is a single vertex, then it is a contracted vertex because it has degree 0. It also is a tree. Therefore F2 has 2 credit. Because F2 results from contracting the blocks of D2, D2 must be a block with 2 credit. Thus the edges of D2 cost $\leq \frac{7}{4}$ opt $(G) - 2$ and for a 2-edge-connected spanning subgraph, as desired. □

5.1 Creating D2

5.1.1 Lemma. *The three steps below create a D2 in polytime*

Proof. There are only 3 steps and each is polytime, therefore it suffices to prove the 3 steps create a D2. A minimal 2-edge cover has the property, that all vertices have degree ≥ 2, and step 2 removes all its bad triangles. Finally the credit we add will have cost $\frac{3}{4}$ opt (G), giving our D2 a cost of $\frac{7}{4}$ opt (G).

□

5.1.1 Calculate a minimal 2-edge cover

We calculate a minimal 2-edge cover, which we can do in polytime. Its cost is a lower-bound on opt (G) because all vertices in a 2ECSS have degree at-least 2.

5.1.2 Handle Bad-Triangles

We handle any bad-triangles in our minimal 2-edge cover.

5.1.2 Lemma. *We can remove all bad triangles from a minimal 2-edge cover in polytime.*

Proof. Proof by contradiction. Suppose any minimal 2-edge cover we can find in polytime has at-least $b > 0$ bad-triangles. Therefore let xyz be a bad triangle. Without loss of generality, let yz be a zero-edge and thus x has an adjacent zero-edge. This is a MAP6 instance, therefore x is not a cut-node. Without loss of generality, let y be adjacent to some $w \neq x, y, z$. Therefore if we swap edge xy for wy, which maintains the 2-edge cover property, thus we get a minimal 2-edge cover with one less $b - 1$ bad-triangles, completing the proof.

□

5.1.3 Add Credit

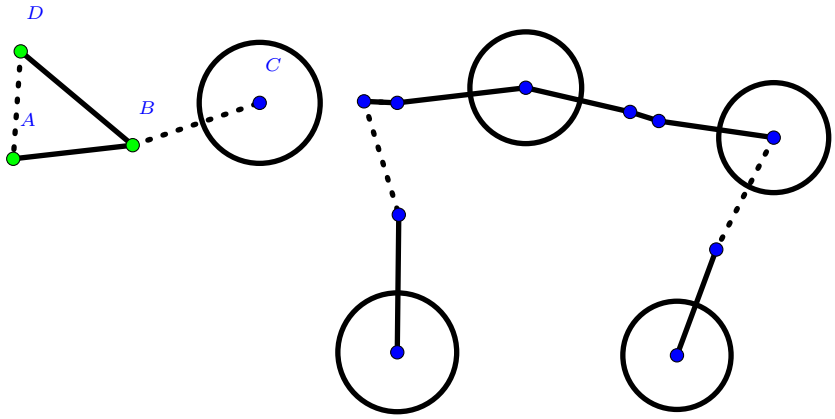
We add credit to our minimal 2-edge cover. For each unit-edge, we add $\frac{3}{8}$ credit to each of the vertices. This credit and the minimal 2-edge cover together form D2. As a result, the cost of D2 is a lower-bound on $\frac{7}{4}$ opt (G). We can add this credit in linear time.

5.2 Creating F2

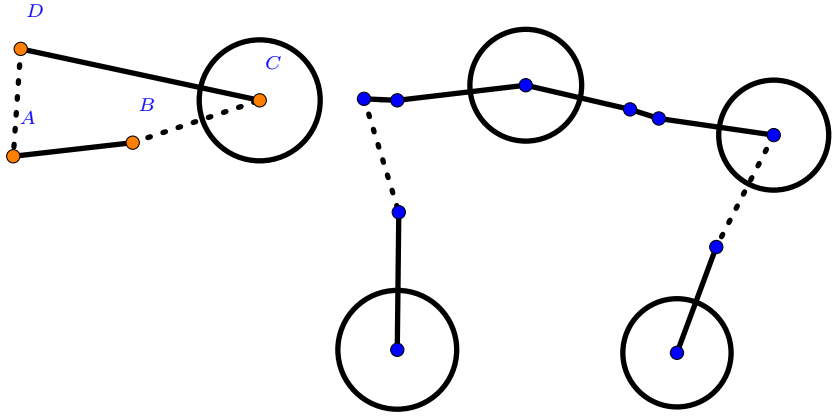
5.2.1 Lemma. *Given a D2, the two steps below create an F2 in polytime*

Proof. There are only 2 steps and each is polytime, therefore it suffices to prove the 2 steps create an F2. Step 1 contracts the blocks of D2 and makes it a forest. Step 2 proves we can assign the credit of D2 to satisfy the credit invariants of F2, which completes the proof.

□



(a) Search for a bad-triangle

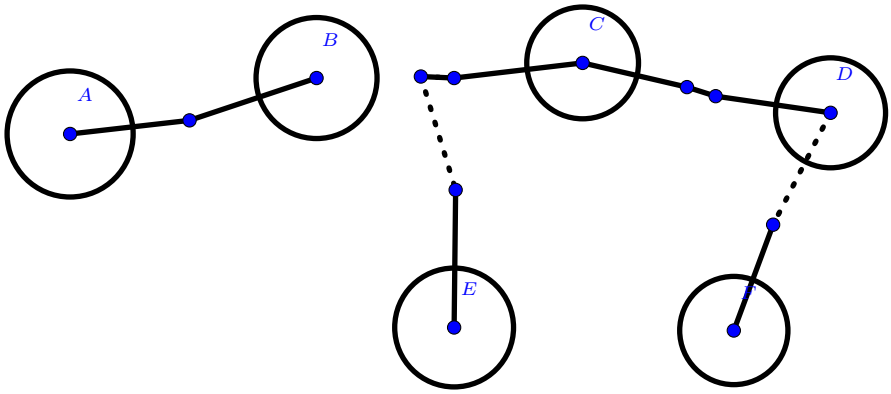


(b) Swap a unit edge in the bad-triangle for another edge

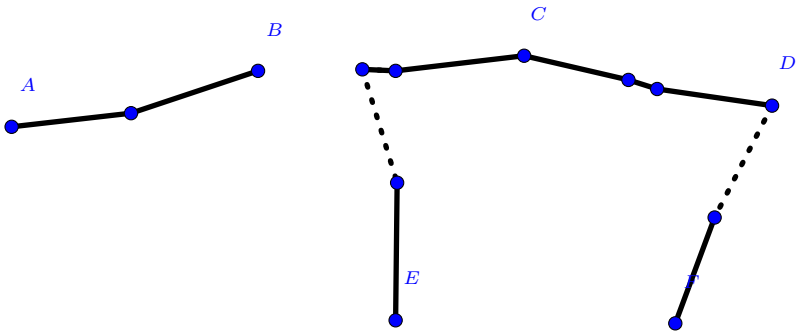
Figure 5.1: The above figure demonstrates Handling Bad-Triangles. Unit-edges are drawn as solid lines, zero-edges are drawn as dotted lines, and large circles represent blocks. The bad triangle ABD (green) is replaced by a square $ABCD$ (orange)

5.2.1 Contract Blocks

We contract all the blocks of D2, which we can do in polytime. Contracting all blocks of a graph creates a forest, because all cycles are blocks.



(a) Select all maximal 2-edge-connected components.



(b) Contract all maximal 2-edge-connected components.

Figure 5.2: The above figure demonstrates Block Contraction. Unit-edges are drawn as solid lines, zero-edges are drawn as dotted lines, and large circles represent blocks. The blocks A, B, C, D, E, F are turned into contracted vertices

5.2.2 Establish credit invariants

Credit Invariants of F2:

1. All trees have 1 credit.
2. All special vertices have .5 credit.
3. All non-special contracted vertices have 1 credit.
4. All uncontracted vertices have $\frac{3}{8}$ credit per adjacent unit edge.

All vertices of D2 are given $\frac{3}{8}$ credit per adjacent unit edge. This is a MAP6 instance, meaning there are no unit multi-edges, thus all blocks have at-least 2 unit edges, meaning they can claim 1.5 credit. Therefore, if a tree has at-least 2 contracted vertices, it can claim .5 credit from each to gain 1 credit while leaving 1 credit for each contracted vertex. The only trees that don't have 2 contracted vertices are those with 1 leaf. When this is the case, the tree claims 1 credit from that contracted vertex, leaving the leaf with 1 credit except when its associated block has only 2 unit edges. These exceptions are all special vertices by definition. They started with 1.5 credit and are left with the .5 credit needed. Therefore with this assignment all credit invariants are satisfied.

Chapter 6

Bridge covering

At the start of this section, we have D2, a spanning subgraph of G with cost $\leq \frac{7}{4}$ opt (G) in which all vertices have degree ≥ 2 and there are no bad-triangles, and F2, a minor of G created by contracting the blocks of D2 (which makes it a forest) and assigning its credit as follows: All trees of F2 have 1 credit, all special vertices of F2 have .5 credit, all other contracted vertices have 1 credit, all uncontracted vertices have $\frac{3}{8}$ credit for each adjacent unit edge.

At the end of this section, we'll have a new D2 that also has no bridges, and thus a new F2 which also has no edges. One thing to note is that any time we alter F2 by adding/removing any edges or credit, we do the same to D2. The only differences between D2 and F2 is that blocks in F2 are contracted and we assign its credit to vertices instead of the graph.

6.0.1 Lemma. *If have a polytime algorithm to find an F2 which is a single vertex given an F2 without edges, then we have a polytime algorithm to find an F2 which is a single vertex.*

Proof. If we repeatedly apply the lemma below at most n times to any given F2, it will become an F2 without bridges. Then we can apply our polytime algorithm to find an F2 which is a single vertex in polytime as desired. □

6.1 Bridge Covering Algorithm

6.1.1 Lemma. *Given an F2 with $b > 0$ bridges, we can return an F2 with at-most $b - 1$ bridges in polytime.*

Proof. Select a tree of F2 with bridges T. Select a leaf l in T. For each vertex v in T, let P_v be the path from v to l in T.

Next we need to select a vertex $u \in T$ such that:

1. There is a path from u to l in $G' - E(F2)$
2. P_u satisfies one of the following:

(a) P_u contains ≥ 2 contracted vertices

(b) P_u has ≥ 3 edges

(c) P_u has exactly 2 edges and P_u contains a vertex, (possibly one of its end vertices) which has ≥ 2 adjacent unit edges in T.

6.1.2 Claim. *A vertex u with the above properties always exists for any choice of leaf l.*

Proof. Suppose not for the sake of contradiction. $\exists u \neq l \in S$ because G is 2-edge-connected. Therefore let $u \in S$. By our assumption, the only contracted vertex in P_u is l. This means we can choose u so that $|P_u| \geq 2$ because G has no cut-nodes. By our assumption, $|P_u| = 2$. Let $v \neq u, l$ be the other vertex in P_u. By our assumption, P_u has no vertex with 2 adjacent unit edges, therefore P_u contains a zero-edge and v has degree 2 in T. vu cannot be a zero-edge because it would be a split, thus vl is a zero-edge. u is not a leaf because it is not a contracted vertex by our assumption. u cannot be adjacent to another unit-edge by our assumption. Therefore uv is a unit-edge with 2 adjacent zero-edges whose contraction creates a cut-node. The zero-edges are in different biconnected components of this cut-node, and two components contain a zero-edge and a leaf, which means they'll have opt ≥ 3. Thus uv is a unit-split, which is a contradiction. □

Therefore there is always a suitable u. The conditions for a suitable u can be checked in polytime, therefore we need only check all vertices of T to find u. Once we have u, choose a path from u to l distinct from T that uses the fewest edges of $G \setminus F2$. We'll call this path Q_u. The shortest path algorithm can give use a suitable Q_u in polytime.

6.1.3 Claim. *Trading $|Q_u|$ credit to add the edges of Q_u to F2 (and D2) and contracting the resulting cycle maintains our credit invariants.*

Proof. There are the same number of edges of $G \setminus D2$ in Q_u as there are trees of F2 that intersect Q_u. Therefore to add the edges of Q_u, we use the 1 credit of each tree involved. After the contraction, these trees become 1 tree in F2. We give the credit from l to this tree to ensure that it still has 1 credit. All that remains is to show that the new contracted vertex has 1 credit.

In the case that P_u has 2 contracted vertices, it has a contracted vertex besides l, with gives ≥ 1 credit to the new contracted vertex.

In the case that P_u has > 2 edges, it has 3 vertices besides l, thus each gives $\geq \frac{3}{8}$ credit to the new contracted vertex.

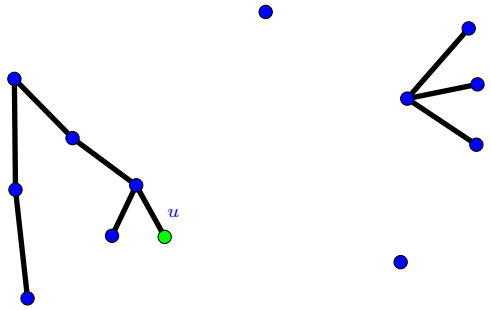
In the case that P_u has a vertex with 2 adjacent unit edges, it has 2 vertices besides l, one gives $\geq \frac{3}{8}$ credit and the other $\geq \frac{6}{8}$ credit to the new contracted vertex.

Therefore, in all cases the new contracted vertex has 1 credit, thereby preserving the credit invariants.

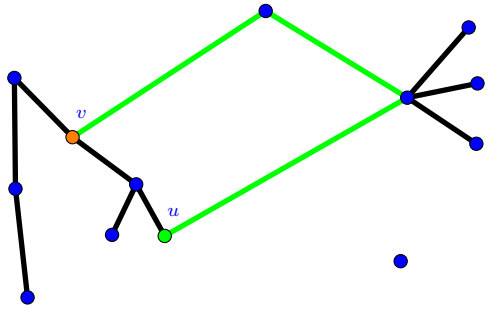
□

Because we swapped credit for edges, we did not increase the cost of D2. We didn't delete any edges, therefore all vertices still have degree ≥ 2. Finally, no bad triangles are created because the only way to add a bad triangle to a D2 by adding edges is to add a zero-edge. This and the fact that credit invariants are maintained mean that we have a new D2 and F2. All edges we added to F2 were contracted along with some bridges in the path P_u, therefore our new F2 has at most $b - 1$ bridges, as desired

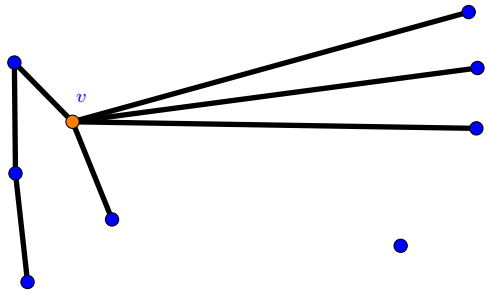
□



(a) Select a vertex v (green) in a tree T with bridges.



(b) Add a path Q (green) in $G' \setminus T$ from v to a suitable vertex u (orange)



(c) Contract the cycle created by adding Q to F2

Figure 6.1: The above figure demonstrates a typical Bridge Covering Subroutine. Edges of F2 are drawn in black. The green path is the path added to F2 to "cover" the bridges between u (green) and v (orange).

Chapter 7

The gluing step

At the start of this section, we have D2, a spanning subgraph of G with cost $\leq \frac{7}{4}$ opt (G) in which all vertices have degree ≥ 2 and there are no bridges or bad-triangles, and F2, a minor of G created by contracting the blocks of D2 (which makes it a forest) and assigning its credit as follows: All trees of F2 have 1 credit, all special vertices of F2 have .5 credit, all other contracted vertices have 1 credit, all uncontracted vertices have $\frac{3}{8}$ credit for each adjacent unit edge.

At the end of this section, we'll have an F2 which is a single vertex, and thus our D2 will be a 2ECSS.

7.0.1 Lemma. *Given an F2 without edges, we have a polytime algorithm to find an F2 which is a single vertex.*

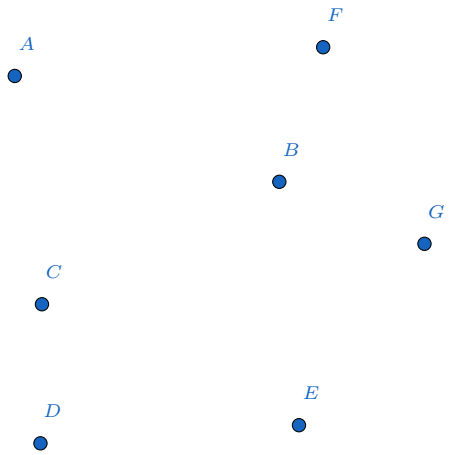
Proof. By repeatedly applying lemma 7.1.4, we can find an F2 such that the graph has no cycles of length ≥ 2, which means F2 is a single vertex. □

7.1 Gluing Algorithm

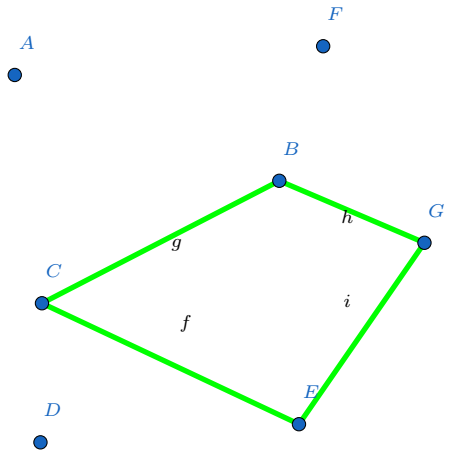
7.1.1 Lemma. *Given a bridgeless F2 in a graph with $b > 0$ cycles of length ≥ 4, we can alter bridgeless F2 so the graph has at most $b - 1$ cycles of length ≥ 4 in polytime.*

Proof. We can find $c \geq 4$ vertices which form a simple cycle C in polytime. Then we can add the c edges of C to F2 (and D2) by swapping 1 credit for each of the c trees of F2. We contract C in F2 in polytime to get a new contracted vertex which is its own tree. There were at-least 4 contracted vertices in C, meaning their credit sums to ≥ 2. We can give 1 credit to the tree, while the contracted vertex retains ≥ 1 credit, thereby maintaining credit invariants. Therefore our new graph is a valid F2, and the graph has at most $b - 1$ cycles of length ≥ 4. □

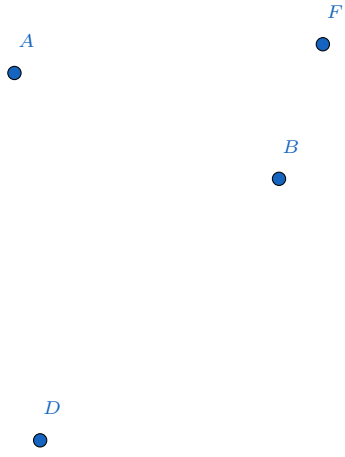
7.1.2 Lemma. *Given a bridgeless F2 in a graph with $b > 0$ cycles of length ≥ 3, we can alter bridgeless F2 so the graph has at most $b - 1$ cycles of length ≥ 3 in polytime.*



(a) Search for a suitable cycle in the graph



(b) Add the edges of the cycle



(c) Contract the cycle created in F2

Figure 7.1: The above figure demonstrates a typical Gluing Subroutine. The green cycle is the cycle added to F2 to "glue" the blocks B, G, E, C together.

Proof. If there are any cycles if length ≥ 4, the result follows from the above lemma. Therefore assume the graph has no cycles of length ≥ 4. We can find 3 vertices which form a simple cycle $C = xyz$ in polytime.

7.1.3 Claim. *This cycle must contain a non-special vertex*

Proof. First note that one vertex x in C has a neighbour w outside C, as otherwise the graph is just 3 special vertices, which would mean G has < 20 vertices, a contradiction. Next note that there exists a path P from w to another vertex of C which does not contain x, since otherwise x would be a split cycle in G. Without loss of generality let P be from w to y. But then Pzx is a cycle of length ≥ 4, a contradiction. □

Then we can add the 3 edges of C to F2 (and D2) by swapping 1 credit for each of the 3 trees of F2. We contract C in F2 in polytime to get a new contracted vertex which is its own tree. There was a non-special contracted vertex in C so we give its 1 credit to the tree, leaving 1 credit from the two other contracted vertices for the new contracted vertex, thereby maintaining credit invariants. Therefore our new graph is a valid F2, and the graph has at most $b - 1$ cycles of length ≥ 3.

□

7.1.4 Lemma. *Given a bridgeless F2 in a graph with $b > 0$ cycles of length ≥ 2, we can alter bridgeless F2 so the graph has at most $b - 1$ cycles of length ≥ 2 in polytime.*

Proof. If there are any cycles if length ≥ 3, the result follows from the above lemma. Therefore assume the graph has no cycles of length ≥ 3. If there are two adjacent non-special vertices, there are two edges between them because the graph is 2-edge connected. We add those two edges to F2 (and D2) by swapping 1 credit for each of the 2 trees of F2. We contract the 2-cycle in F2 in polytime to get a new contracted vertex which is its own tree. Both vertices had 1 credit, therefore we give 1 credit to the tree and 1 to the vertex, thereby maintaining credit invariants. Therefore our new graph is a valid F2, and the graph has at most $b - 1$ cycles of length ≥ 2. We can therefore assume there are no adjacent non-special vertices.

Note that graphs without cycles of length > 2 have a tree-like structure. A special vertex cannot be an internal node of this tree, as that would mean there was a split cycle in G. Therefore, because there are no adjacent non-special vertices, this tree consists of special vertices which are all adjacent to one non-special vertex r and nothing else.

Select special vertex v. If we uncontract v, we get a block B which is a triangle xyz with a zero-edge, or a square $wxyz$ with two zero-edges. In both cases, there are no cut-nodes in G, therefore two vertices of B are adjacent to r. In both cases, let yz be a zero-edge and let z be adjacent to r without loss of generality. Suppose $B = xyz$. There are no splits, so x is adjacent to r. Swap xz for xr and use the credit from v to buy edgezr. This creates the block $rxyz$, which we contract. We have not used any credit from r, therefore the credit

invariants remain intact. Now suppose $B = wxyz$. If 3 vertices are adjacent to r, swap a unit-edge and 1 credit for 2 unit-edges adjacent to r in the same way as before. B is not a redundant-cycle, therefore if only two vertices are adjacent to r, then the other two are adjacent. Also, there are no splits, therefore one of w, x is adjacent to r. Therefore, without loss of generality, we can assume w is adjacent to r, because otherwise $xwyz$ is also a square. Swap wz for xr and use the credit from v to buy edgezr. This creates the block $rwxyz$, which we contract. We have not used any credit from r, therefore the credit invariants remain intact. Therefore our new graph is a valid F2, and the graph has at most $b - 1$ cycles of length ≥ 2.

□

Chapter 8

Conclusion

In this final chapter, we use the results of previous chapters to give a quick proof of the main result in this paper.

8.0.1 Theorem. *There is a $\frac{7}{4}$ polytime approximation algorithm for MAP*

Proof. Either a MAP instance is small enough to solved optimally, or large enough to be reduced to a set of MAP6 instances in polytime via lemmas 4.1.5, 4.2.5, 4.3.5, 4.4.5, 4.5.5 and 4.6.5. The 2ECSSs we return for each MAP6 instance, along with any edges contracted when we separate the instances, will form a 2ECSS for the original instance. Lemmas 5.01, 6.0.1, and 7.0.1 show that there is a $\frac{7}{4}$ polytime approximation algorithm for MAP6, and thus for MAP. □

Bibliography

[1] D. Adjiashvili. Beating approximation factor two for weighted tree augmentation with bounded costs. In: P. N. Klein (ed.) *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 2384–2399. SIAM, 2017.

[2] R. Diestel. *Graph Theory (4th ed.)*. Graduate Texts in Mathematics, Volume 173. Springer-Verlag, Heidelberg, 2010.

[3] G. Even, J. Feldman, G. Kortsarz, and Z. Nutov. A 1.8 approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. *ACM Trans. Algorithms*, 5(2):21:1–17, 2009.

[4] S. Fiorini, M. Groß, J. Könemann, and L. Sanità. Approximating weighted tree augmentation via Chvátal-Gomory cuts, In: A. Czumaj (ed.) *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 817–831. SIAM, 2018.

[5] G. N. Frederickson and J. JáJá. Approximation algorithms for several graph augmentation problems. *SIAM J. Comput.*, 10(2):270–283, 1981.

[6] G. N. Frederickson and J. JáJá. On the relationship between the biconnectivity augmentation and traveling salesman problems. *Theor. Comput. Sci.*, 19:189–201, 1982.

[7] H. N. Gabow, M. X. Goemans, É. Tardos, and D. P. Williamson. Approximating the smallest$k$-edge connected spanning subgraph by LP-rounding. *Networks*, 53(4):345–357, 2009.

[8] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.

[9] F. Grandoni, C. Kalaitzis, and R. Zenklusen. Improved approximation for tree augmentation: saving by rewiring. *Proc. 50th ACM Symposium on Theory of Computing, STOC*, pages 632–645, 2018.

[10] K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

[11] M. Karpinski, M. Lampis, and R. Schmied. New inapproximability bounds for TSP. *J. Comput. Syst. Sci.*, 81(8):1665–1677, 2015.

[12] G. Kortsarz and Z. Nutov. A simplified 1.5-approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. *ACM Trans. Algorithms*, 12(2):23:1–20, 2016.

[13] L. C. Lau, R. Ravi, and M. Singh. *Iterative Methods in Combinatorial Optimization*. Cambridge Texts in Applied Mathematics (No. 46). Cambridge University Press, 2011.

[14] L. Lovàsz and M. D. Plummer. *Matching Theory*, volume 367 of *AMS/Chelsea Publishing*. American Mathematical Society, 2009.

[15] H. Nagamochi. An approximation for finding a smallest 2-edge connected subgraph containing a specified spanning tree. *Discrete Applied Mathematics*, 126:83–113, 2003.

[16] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics, Volume 24. Springer-Verlag, Berlin Heidelberg, 2003.

[17] A. Sebö and J. Vygen. Shorter tours by nicer ears: 7/5-approximation for the graph-TSP, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Combinatorica*, 34(5):597–629, 2014.

[18] S. Vempala and A. Vetta. Factor 4/3 approximations for minimum 2-connected subgraphs. In K. Jansen and S. Khuller, (eds.) *Approximation Algorithms for Combinatorial Optimization, Third International Workshop, APPROX 2000, Proceedings*, LNCS 1913, pages 262–273. Springer, 2000.

[19] H. Whitney. Non-separable and planar graphs. *Trans. Amer. Math. Soc.*, 34:339–362, 1932.

[20] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[21] J. Cheriyan, J. Dippel, F. Grandoni, A. Khan, and V. V. Narayan *The Matching Augmentation Problem: A $\frac{7}{4}$-Approximation Algorithm*. arXiv:1810.07816 [cs.DS]