Approaches to approximating the minimum weight k-edge connected spanning subgraph of a mixed graph

by

Dominik Alban Scheder

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirements for the degree of
Master of Science
Department of Computer Science
2006

Abstract

The problem of finding the minimum weight k-edge connected spanning subgraph of a mixed graph is NP-hard for every $k \geq 1$, and, for $k \geq 2$, the best approximation ratio known so far is 4. In this thesis, we analyze several approaches to the general k-ECSS problem of mixed graphs.

We give a factor 2 approximation for a special case in which all undirected edges have the same weight, and this weight is no higher than twice the weight of the cheapest directed edge. For the special case in which undirected edges have no higher weights than directed edges (though not necessarily being uniform) we achieve a factor 3.75 approximation.

Further, we give several examples on which the analyzed algorithms perform poorly.

Acknowledgements

I want to thank all the people who contributed to this work. In particular, I want to thank my advisor Harold Gabow. Further, the German Fulbright Program for funding my first year of study at the University of Colorado.

I am thankful for the patient proof-hearing, proof-reading and advice from Christoph Koutschan, Hans Löhr, Christian Dörr and Stefan Büttcher.

Contents

Figures

Figure

Chapter 1

Introduction

A graph $G = (V, E)$ is called k-edge connected (k-EC) if it contains at least k edge disjoint paths — i. e. paths that do not share an edge — between every two vertices. This is equivalent to say that every set of vertices, except $\emptyset$ and V, is entered by at least k edges. A natural problem is how to find a minimum cost k-edge connected spanning subgraph (k-ECSS) of G, that is, a k-EC spanning subgraph containing the smallest possible number of edges, or having the smallest possible total weight, if the graph is weighted.

One can think about this problem as the task of designing a fault-tolerant communication network between several hosts, which means the network should survive a certain number of link failures, i. e. every host should still be able to communicate with any other.

The problem comes in several flavours: the graph can be directed or undirected, the edges can be weighted or not. Further, we can allow parallel edges (multigraphs) or require all edges to be simple (simple graphs).

The problem of finding the smallest or minimum weight k-ECSS is NP-hard for every $k \geq 2$, and, if the graph is directed, also for $k = 1$. This follows easily from a reduction from Hamiltonian Cycle: consider an unweighted, undirected graph G with n vertices, and let $k = 2$. Every 2-ECSS must contain at least n edges, and there is a 2-ECSS of G containing exactly n edges if and only if G has a Hamiltonian cycle.

Since it is usually assumed that P $\neq$ NP, there is little hope for finding the minimum weight k-ECSS in polynomial time. Therefore people are looking for **approximation algorithms**, i. e. algorithms that run in polynomial time and produce a solution which has a weight not much higher than the weight of an optimal solution.

For a given problem instance I, we say an algorithm achieves a ratio of α on I if it finds a

solution that has a weight which is α times the weight of an optimal solution. If an algorithm achieves a ratio of at most α on every problem instance, we call it an α-approximation algorithm.

If we can prove a ratio of α for some algorithm, and also find a problem instance on which the algorithm achieves a ratio of α, we say the analysis of the algorithm is **tight**, and call the problem instance a **tight example**. Sometimes, we do not come up with a single example, but rather with a family of examples, on which the ratios the algorithm achieves are strictly smaller than α, but do converge to α. We then still call the analysis tight, or, to be more precise, asymptotically tight.

Previous Work

Khuller and Raghavachari [7] give a depth-first search algorithm for the k-ECSS problem on unweighted undirected multigraphs that achieves a ratio of 1.85, which was the first improve over the previously known ratio of 2. Gabow [3] improved the analysis of this algorithm to a factor of 1.61.

For unweighted multigraphs, [5] gives an LP-rounding algorithm achieving an approximation ratio of $1 + 2/k$ for directed graphs or undirected graphs with k even, and $1 + 3/k$ for undirected graphs and k odd. There is also another rounding technique which achieves a ratio of $1 + 2/k$ on undirected graphs for arbitrary k.

For weighted graphs, there has been considerably less progress. An application of Lawler's minimum weight matroid intersection algorithm [8] yields a factor 2 approximation for the k-ECSS problem for both undirected and directed weighted multigraphs. However, this is the best ratio known so far. A far more general problem is the Steiner network problem, where each vertex pair u, v has a **cut-requirement** $r(u, v)$, and we are looking for a minimum weight subgraph that contains at least $r(u, v)$ edge disjoint paths between u and v. Note that setting $r(u, v) = k$ for every vertex pair gives the k-ECSS problem.

Jain [6] gives an iterated LP-rounding algorithm for the Steiner network problem on undirected graphs which achieves a ratio of 2. Gabow [4] extends Jain's proof to directed graphs and achieves a factor of 3, but also shows that iterated rounding has unbounded approximation ratio on mixed graphs.

Our Results

The best currently known approximation algorithm for the minimum weight k-ECSS problem on mixed graphs achieves a ratio of 4. Given a mixed graph G, the algorithm builds the directed version D by replacing each undirected edge by two directed edges, one in each direction. On D, one can run any algorithm for the k-ECSS problem on directed graphs, so e. g. the one using the minimum

weight matroid intersection algorithm cited above. It is easy to show that an algorithm achieving a ratio of α on directed graphs will achieve a factor of 2α on mixed graphs.

In Chapter 3 we show how to achieve the factor 2 approximation on directed and undirected graphs, and how this yields the factor 4 for mixed graphs.

In Chapter 4 we present our results. In Section 4.1, we prove an approximation ratio of 2 for a special case. In this special case, we assume that all undirected edges have uniform weights, and that this weight is at most twice the weight of the cheapest (lowest weight) directed edge.

Section 4.2 gives some examples demonstrating that the simple approach of 4.1 does not work for more general graphs.

In Section 4.3, we will achieve a factor 3.75 approximation for mixed graphs where undirected edges have no higher weights than directed edges (we do not require uniformity here) and $k = 2$. We think this bound is far from being tight, and also think it should generalize to higher k.

In Section 4.4 we give some examples on which the algorithm analyzed in Section 4.3 performs poorly. We did not find tight examples, though.

Finally, in Section 4.5 we analyze another special case, where all directed edges have weight zero and undirected edges have uniform weight cost. We did not succeed in proving any ratio better than 4, but did find some results that might be useful.

Note that we always allow parallel edges, i. e. we are dealing with multigraphs.

Chapter 2

Notation

Consider a directed graph $G = (V, E)$ and a vertex set $X \subset V$, and an edge $e = (u, v) \in E$. The vertex v is called the **head** of e, u is called the **tail** of e. An edge $e = (u, v)$ is said to **enter** X if $v \in X$ but $u \notin X$. If $u \in X$ but $v \notin X$, e **leaves** X. An edge set F is said to enter (leave) X if it contains an edge entering (leaving) X. Further $\rho_F(X)$ and $\delta_F(X)$ denote the number of edges of F entering X and leaving X, respectively.

If G is undirected, an edge $e = \{u, v\}$ is called **adjacent** to X if exactly one of u and v is inside X. In this case, we write $d_F(X)$ to count the edges of F adjacent to X. If G is mixed, i. e. contains both directed and undirected edges, an undirected edge $e = \{u, v\}$ that is adjacent to some set is said to both enter and leave it. Thus, it is counted by ρ as well as by δ.

It is well known that ρ and δ are **submodular**, i. e. they fulfill the inequality

$$\rho(M) + \rho(N) \geq \rho(M \cup N) + \rho(M \cap N)$$

for any vertex sets M and N.

For an edge set F, $V(F)$ is the set of all vertices spanned by F, i. e. the set of all endpoints (heads and tails) of edges in F. If V is a set, we sometimes write $V - v$ and $V + v$ instead of $V \setminus \{v\}$ and $V \cup \{v\}$. Further, given a vertex set X, an edge is called **inside** X if both its endpoints are in X, and **outside** if neither endpoint is in X.

Two sets X, Y are said to **intersect** if $X \cap Y$ is nonempty. They are said to **cross** if none of the sets $X \cap Y$, $X \setminus Y$ and $Y \setminus X$ is empty. This means, two sets cross if they are neither disjoint, nor one is a subset of the other.

For mixed graphs, we have to introduce some additional notation: given a mixed edge set E, we let E^U denote the set of all undirected edges in E, and E^D the set of all directed edges. $\vec{E}$ denotes the **directed version** of E, i. e. it contains all directed edges of E and, for each undirected

edge $e = \{u, v\} \in E$, it contains the two antiparallel directed copies of e, which are (u, v) and (v, u). Remember that we always allow parallel edges.

Given a mixed edge set E and a directed edge set $F \subseteq \vec{E}$, $\bar{F}$ denotes the original mixed version of F, which means it contains the directed edges of F, and includes an undirected edge $\{u, v\}$ if at least one of its antiparallel directed copies is in F.

In F, an edge $e = (u, v)$ that is a copy of an undirected edge in E is called a **single undirected edge** if $(v, u) \notin F$. Otherwise, it is called a **double** undirected edge.[1] F_d^U denotes the set of all double undirected edges in F, F_s^D the set of all single undirected edges. Further, we define $F^U := F_d^U \cup F_s^U$ and $F^D = F \setminus F^U$, so F^D is the set of all edges in F that have a directed original in the mixed set E.

Throughout this paper, A denotes a solution returned by some algorithm (it will be clear from the context which algorithm), and OPT denotes an optimal solution to the problem that is analyzed.

[1] Note that F is a **directed** edge set. So we extend the notion of an **undirected** edge to directed sets that were built from a mixed set

Chapter 3

The Branching Algorithm

We mentioned that Lawler's minimum weight matroid intersection algorithm yields a factor 2 approximation on directed graphs. In this Chapter, we demonstrate how to achieve that and how to apply this algorithm to undirected and mixed graphs. We refer to that algorithm as the **branching algorithm**.

3.1 Directed Graphs

Before starting to analyze algorithms for the k-ECSS problem, we will introduce some additional notation and facts that will be crucial to our analysis.

Definition 3.1 Given a directed graph $G = (V, E)$, a root vertex $r \in V$ and a positive integer k, a set $F \subseteq E$ is called k**-entering** if every set $X \subseteq V - r$ has $\rho(X) \geq k$. Similarly, $F \subseteq E$ is called k**-leaving** if every set $X \subseteq V - r$ has $\delta(X) \geq k$.

We state two key properties of k-entering sets.

Proposition 3.1 F is k-entering (k-leaving) if and only if for every vertex $s \in V - r$, there are $\geq k$ edge-disjoint $r - s$-paths ($s - r$-paths).

This fact is a direct consequence of the well-known Min Cut—Max Flow theorem. The next Proposition is less obvious but equally important:

Proposition 3.2 F is k-entering (k-leaving) if and only if F can be partitioned into k 1-entering (1-leaving) subsets $F_1, \ldots, F_k$.

Note that a 1-entering set F always contains a spanning arborescence T rooted at r.[1] We call T an out-tree rooted at r, or simply an out-tree, if there is no doubt about r. Every minimal

[1] an **arborescence** is simply a directed tree in which either all edges are directed away from the root or all edges are directed towards the root. We will use the words tree and arborescence as synonyms

1-entering set is an out-tree, and every minimal k-entering set F can be partitioned into k out-trees. In this case, we call F a k-out-tree (rooted at r). Similarly, a minimal k-leaving set is called a k-in-tree.

There are several proofs of Proposition 3.2, the first one given by Edmonds [1]. In this thesis, we will look at a proof by Frank [2], because it is very clear and gives us a polynomial algorithm to construct the decomposition. He proves a more general result about so-called **Kernel systems**, of which the above Proposition is a special case. To make this paper self-contained, we will restate Frank's proof here, adapted to our special case.

Proof of Proposition 3.2. The "if"-direction is trivial. For the "only if"-direction, we will show how to partition F into a 1-entering set F_1 and a $k-1$-entering set F'. Repeating this process proves the proposition.

We start with F_1 empty and $F' = F$. We then iteratively remove an edge from F' and add it to F_1. This is repeated until F_1 is 1-entering. The difficulty is to assure that F' stays $k-1$-entering all the time.

At any stage of the algorithm, a set $X \subseteq V - r$ is called **dangerous** if $\rho_{F'}(X) = k-1$. This means, an edge $e \in F'$ that enters X must not be moved to F_1, since otherwise F' would not be $k-1$-entering any more. A set $X \subseteq V - r$ is called **unsatisfied** if F_1 does not enter X.

In every step, let X be some maximal unsatisfied set. We find some edge $e \in F'$ that does not enter any dangerous set, remove it from F' and insert it into F_1 and say e was inserted into F_1 **because of** X. Clearly, this keeps F' $k-1$-entering.

We only have to show that such an edge e always exist. In the following, F_1 and F' always denote the sets under construction just before edge e is removed from F' and added to F_1.

Proposition 3.3 Let X be any maximal unsatisfied set. For any $f \in F_1$, the head of f is not in X.

Proof. Let $f = (u, v)$. Suppose for the sake of contradiction that $v \in X$. Since X is unsatisfied, f does not enter X, so we must also have $u \in X$. Let F_1' be the set of edges the algorithm has inserted into F_1 before f, and let X_f be the set because of which f was inserted. Obviously, F_1' enters neither X_f nor X. Hence it neither enters $X \cup X_f$, so this set must be unsatisfied, too. Also, $X \cup X_f$ is larger than X_f (it contains u, which X_f does not contain), which contradicts the fact that the algorithm chooses the maximal unsatisfied set in each step. □

Proposition 3.4 If X_D is a dangerous set, then it is not a subset of X.

Proof. A dangerous set cannot be unsatisfied. Therefore, there is some edge $f \in F_1$ entering X_D. So the head of f is in X_D, but, by Proposition 3.3, not in X. □

Proposition 3.5 If X and Y are dangerous and $X \cap Y \neq \emptyset$, then $X \cap Y$ is also dangerous.

Proof. Since ρ is a submodular function, we have

$$(k-1) + (k-1) = \rho_{F'}(X) + \rho_{F'}(Y) \geq \rho_{F'}(X \cup Y) + \rho_{F'}(X \cap Y) \geq (k-1) + (k-1)$$

This implies that every inequality must hold with equality. Since $X \cap Y \neq \emptyset$, we have $\rho_{F'}(X \cap Y) \geq k-1$, and since $X \cup Y \neq V$ (neither set contains the root r), we have $\rho_{F'}(X \cup Y) \geq k-1$ as well, so both $X \cup Y$ and $X \cap Y$ must be dangerous. □

Now, for the sake of contradiction, suppose that every edge $e = (u, v) \in F'$ that enters X also enters some dangerous set. Such a dangerous set must intersect with X, since both sets contain v. Let X_D be a dangerous set intersecting with X and $X_D - X$ as small as possible. $X_D - X$ cannot be empty, since otherwise $X_D \subset X$, contradicting Proposition 3.4.

Now, we show that there is some edge $e = (u, v) \in F'$ with $u \in X_D - X$ and $v \in X_D \cap X$. Suppose that no such edge exists, then every edge entering $X_D \cap X$ also enters X_D. Since exactly $k-1$ edges enter X_D, $X_D \cap X$ is also dangerous. But it is a subset of X, contradicting Proposition 3.4. So there is an edge $e = (u, v) \in F'$ with $u \in X_D - X$ and $v \in X_D \cap X$.

We claim that edge e does not enter any dangerous set. Suppose it did enter the dangerous set X_e. Then $X' = X_e \cap X_D$ is nonempty (both sets contain vertex v), hence also dangerous by Proposition 3.5. But $X' - X$ is smaller than $X_D - X$, because it does not contain u, which X_D does contain, so this contradicts the fact that X_D was chosen such that $X_D - X$ is as small as possible.

This means there is some edge $e \in F'$ that enters X but no dangerous set, so e can safely be moved from F' to F_1. This proves Proposition 3.2. □

It is interesting to ask what F_1 will look like at each stage of the algorithm.

Proposition 3.6 At any stage of the algorithm, F_1 is a subtree of F, rooted at r.

Proof. We prove this by induction. At the beginning, F_1 is empty, hence trivially a subtree. For simplicity we say at the beginning F_1 spans only the root r, i. e. $V(F_1) = \{r\}$. Now suppose F_1 is a subtree rooted at r, but not a spanning subtree (if it is spanning, F_1 is also 1-entering, hence we are done). Then the maximal unsatisfied set X cannot contain any vertex spanned by F_1. Clearly,

$V - V(F_1)$, the set of all vertices not spanned by F_1 is nonempty, since F_1 is not spanning yet. Also, $V - V(F_1)$ is unsatisfied, since no edge of F_1 enters it. It is indeed the unique maximal unsatisfied set. To see this, first note that no unsatisfied set X can contain a vertex in $V - V(F_1)$, otherwise there would be at least one edge of F_1 entering X (since $r \notin X$). Also, if X is unsatisfied and $v \in V - V(F_1)$, then $X + v$ is unsatisfied as well, since no edge of F_1 enters v.

Now, the algorithm will add an edge $e \in F'$ to F_1 which enters X, so $F_1 + e$ is still a tree rooted at r. □

Note that Frank's proof gives us a polynomial time algorithm to construct the partition of a k-entering set F into k 1-entering sets. This follows because it is easy to find a maximal unsatisfied set, as stated in Proposition 3.6, and also possible, in polynomial time, to check whether an edge $e \in F'$ enters a dangerous set: if it does, then F' is not $k-1$-entering any more, so there is a vertex s such that F' contains less than $k-1$ $r-s$-paths. To check this, we have to find the minimum $r-s$-cut for every vertex s.

Given a weighted graph $G = (V, E)$, $w : E \to \mathbf{R}_{\mathbf{0}}^{+}$, an interesting question is whether one can find a **minimum weight k-entering set** F in polynomial time The weighted matroid intersection algorithm first given by E. L. Lawler [8] achieves that. Since such an F is minimal, we call it a **minimum weight k-out-tree**.

This immediately yields a factor-2 approximation for the minimum weight k-ECSS problem of directed graphs:

Lemma 3.1 Given an integer k and a weighted k-edge connected graph $G = (V, E)$. Choose $r \in V$ arbitrarily. Let O be a minimum weight k-out-tree and I a minimum weight k-in-tree, both rooted at r. Then $H := I \cup O$ is k-EC and its weight is at most twice the weight of a minimum weight k-ECSS of G.

Proof. First we show that H is k-EC. Consider any proper subset X of V. If X does not contain r, then $\rho_H(X) \geq k$, since H contains O, and O is a k-out-tree. If X does contain r, than $Y := V - X$ does not contain r. Since H contains the k-in-tree I, we have $\delta_H(Y) \geq k$. Of course, $\rho_H(X) = \delta_H(Y)$. Hence any set $X \subseteq V$ has an in-degree of at least k.

To prove the approximation ratio, fix any optimal k-ECSS OPT. Of course, OPT is both k-entering and k-leaving. This means $w(OPT) \geq w(O)$ and $w(OPT) \geq w(I)$, hence we have $w(H) \leq w(I) + w(O) \leq 2w(OPT)$. □

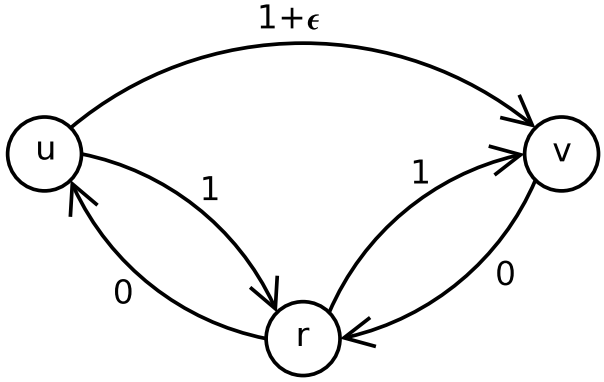

Figure 3.1: Tight example for the factor-2 approximation algorithm on directed graphs with $k = 1$

This is currently the best known result, for any $k \geq 1$, and the bound is tight for every k. We give a tight example for $k = 1$ which can easily be generalized for higher k.

In Figure 3.1, the minimum weight 1-out-tree contains (r, u) and (r, v) and has weight 1. The minimum weight 1-in-tree contains (u, r) and (v, r) and also has weight 1. The optimal 1-ECSS (strongly connected spanning subgraph, in that case), is $\{(r, u), (u, v), (v, r)\}$ and has weight $1 + \epsilon$, with ϵ being an arbitrarily small positive real number. So the approximation ratio is arbitrarily close to 2. This example easily generalizes to k by replacing each edge by k parallel copies.

3.2 Undirected Graphs

For the k-ECSS problem on undirected graphs, a slightly different algorithm works, but also here a factor 2 is the best we know at this time. Of course, for $k = 1$, the k-ECSS problem on undirected graphs is simply the minimum spanning tree problem, hence solvable in polynomial time. For $k \geq 2$ though, the problem remains NP-hard also for undirected graphs.

Let $G = (V, E)$ be a k-EC undirected graph. To achieve a factor-2 approximation, build $D = \vec{G}$, the directed version of G. Further, set $w(u, v) = w(v, u) = w(\{u, v\})$. On D, choose the root vertex r arbitrarily and find the minimum weight k-out-tree O rooted at r. Now, $\bar{O} \subseteq E$ is the set that includes each edge of which O contains at least one direction.

Lemma 3.2 The undirected graph $(V, \bar{O})$ is k-EC and its weight is at most twice the weight of the optimal k-ECSS of G.

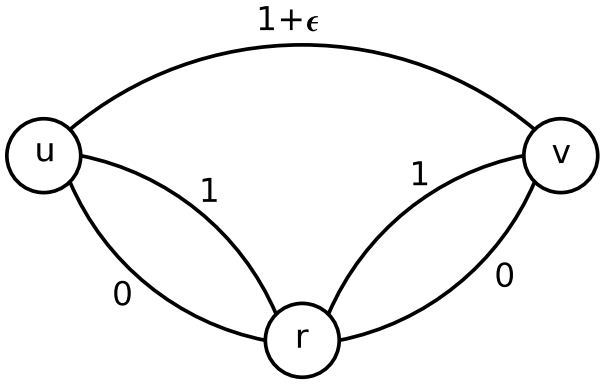

Figure 3.2: Tight example for the factor-2 approximation algorithm on undirected graphs with $k = 2$

Proof. Since G is undirected, we do not distinguish between ρ and δ. Consider any set $X \subset V$. If X does not contain the root, then at least k edges of $\bar{O}$ must be adjacent to X, since at least k edges of O enter X. If X contains the root, the same argument holds for $V - X$, so $V - X$ is adjacent to at least k edges of $\bar{O}$. But an edge adjacent to $V - X$ is of course also adjacent to X. So $(V, \bar{O})$ is a k-ECSS of G.

To prove the approximation factor, let $O\vec{P}T$ be the directed version of OPT. $O\vec{P}T_D$ is a k-ECSS of D, hence is k-entering. Therefore we have $w(O) \leq w(O\vec{P}T)$. Since $O\vec{P}T$ contains two antiparallel copies for each edge in OPT, we have $w(O\vec{P}T) = 2w(OPT)$. The approximation ratio follows. □

Also for this case we have a tight example. It is almost identical with the example for directed graphs. Here, we have $k = 2$.

Note that the graph in Figure 3.2 has parallel edges. Our algorithm replaces each edge by two antiparallel directed edges and finds the optimal 2-out-tree O, which is $\{(r,u)_1, (r,u)_2, (r,v)_1, (r,v)_2\}$, where we write $(r,u)_i$ to indicate that there are parallel edges between these vertices. The weight of O is 2. The optimal 2-ECSS of course is $\{\{r,u\}_1, \{u,v\}, \{v,r\}_1\}$ and has weight $1 + \epsilon$. Hence, $O\vec{P}T$ has weight $2 + 2\epsilon$, slightly more expensive than O.

If one is uncomfortable with this example because it contains parallel edges, there is an example without parallel edges, having uniform edge costs, on which the algorithm achieves a factor 2, too: let $V = \{u_0, \ldots, u_{n-1}\}$, n odd, and let $E = C_1 \cup C_2 := \{\ \{u_i, u_{i+1 \mod n}\},\ 0 \leq i < n\} \cup \{\ \{u_i, u_{i+2 \mod n}\},\ 0 \leq i < n\}$, so $G = (V, E)$ has two edge disjoint Hamiltonian cycles C_1, C_2. For $k = 2$, the optimal k-ECSS is simply a Hamiltonian cycle, hence has weight n, whereas we can assume that our algorithm is so unlucky that it chooses $O = C_1 - \{u_{n-1}, u_0\} \cup C_2 - \{u_{n-2}, u_0\}$, which has a weight of $2n - 2$.

We state here that while we can "simulate" undirected graphs by directed graphs by replacing undirected edges by antiparallel copies, this can change the approximation ratio.

In fact, the problem of finding a minimum weight k-entering set is solvable in polynomial time on directed graphs, but NP-hard on undirected graphs. On the other hand, the k-ECSS requires us to construct a k-entering set as well as a k-leaving set in the directed case, whereas on undirected graphs, we only have to construct one of these sets. A generalization that contains directed and undirected graphs as special cases are mixed graphs, i. e. graphs in which some edges are directed and some are undirected.

3.3 Mixed Graphs

Now we come to the central problem of this thesis, which is how to approximate the minimum weight k-ECSS of a mixed graph. Finding a minimum weight k-out-tree of an undirected graph is NP-hard, but on the other hand we do not have to find an additional k-in-tree in the undirected case, since every out-tree works also as in-tree.

On directed graphs, the k-out-tree can be constructed optimally, but we do have to find an additional k-in-tree. Thus, in the first case we have to solve one problem once, with an approximation ratio of 2. In the latter case, we have to solve a problem twice, but with an approximation ratio of 1.

On mixed graphs, both disadvantages are present, and we must construct the k-out-tree **and** the k-in-tree, both with an approximation ratio of 2, in the worst case. This yields an approximation ratio of 4. We describe the algorithm more formally:

Given a mixed graph $G = (V, E)$, construct the directed version $\vec{G}$ of G by replacing undirected edges by two antiparallel directed copies having the same weight as the original edge, and leave directed edges in G unchanged. On $\vec{G}$, we construct O and I, the minimum weight k-out-tree and k-in-tree. Return $A = \overline{O \cup I}$, the original mixed version of $O \cup I$. We refer to this algorithm as the **branching algorithm**, because it does nothing more than constructing the out-branchings and in-branchings O and I.

Lemma 3.3 *A is k-EC and its weight is at most four times the weight of the optimal k-ECSS.*

Proof. The proof that A is k-EC is quite similar to the proofs above. So we only prove the approximation ratio. We have $w(O\vec{P}T) \leq 2w(OPT)$, since each directed edge in OPT contributes the same to $w(O\vec{P}T)$ as to $w(OPT)$, and each undirected edge in OPT contributes twice as much. As

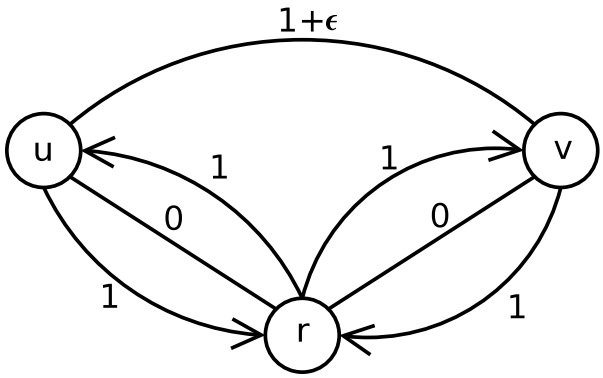

Figure 3.3: Tight example for the factor-4 approximation algorithm on mixed graphs, for $k = 2$. Edges with arrows are directed, edges without arrows are undirected

usual, we have $w(O), w(I) \leq w(O\vec{P}T)$. Adding yields

$$w(A) \leq w(\bar{O}) + w(\bar{I}) \leq w(O) + w(I) \leq 2w(O\vec{P}T) \leq 4w(OPT)$$

Which is the approximation ratio of 4. □

The proof suggests how a tight example should look like: For the first inequality to come close to equality, I and O should be disjoint, or have only very cheap (low-weight) edges in common. For the second, neither set should include many antiparallel copies of undirected edges, while the forth says OPT should have many of them. Indeed such an example exists (see Figure 3.3). Not surprisingly, it is very similar to those we have seen before.

Chapter 4

Adapting the Weight Function

The previous example gives rise to an important question: the worst case of a ratio of 4 can only be achieved if O and I use only one direction of an undirected edge (i. e. include only one of its two antiparallel copies) and OPT uses many undirected edges in both directions.

This last point needs clarification. OPT is a mixed edge set, so it is not well-defined which undirected edge is used in both or only in one direction (O and I are subgraphs of the directed graph $\vec{G}$, so here it is well-defined). We define that as follows: $O\vec{P}T$ is a k-ECSS of $\vec{G}$, so it contains a k-out-tree O_{OPT} and a k-in-tree I_{OPT} rooted at the vertex r our algorithm has chosen as root vertex. These k-trees do not need to be unique, but if we fix some O_{OPT} and I_{OPT} arbitrarily, we can count how often OPT uses an undirected edge e. An undirected edge can be used up to four times (remember that the in-tree and the out-tree need not be edge-disjoint).

The example in Figure 3.3 is pathologically bad because the algorithm fails to use the very valuable undirected edge $\{u, v\}$. It might now seem that our choice to set $w(u, v)$ and $w(v, u)$ in $\vec{G}$ equal to $w(\{u, v\})$, the weight of the original undirected edge, was not optimal. The question is whether we can improve the approximation ratio by using an **adapted weight function** w', $w'(u, v) = w'(v, u) = \gamma w(\{u, v\})$, for some $\gamma < 1$. Unfortunately, the answer is no, for the time being, but the approximation ratio does not increase, either:

Lemma 4.1 Let $G = (V, E)$ be some mixed graph with edge weight function w. If we use an adapted weight function w' for any $1/2 \leq \gamma \leq 1$, the branching algorithm achieves an approximation ratio of 4.

Proof. Two inequalities change: Before, we had $w(A) \leq w(I \cup O)$. This is no longer true, since for an undirected edge $\{u, v\}$, we have $w(\{u, v\}) = \frac{1}{\gamma} w'(u, v)$. So the weight might inflate by $1/\gamma$ when

we build the mixed set A from the directed set $I \cup O$, and we have

$$w(A) \leq \frac{1}{\gamma} w'(I \cup O)$$

On the other hand, we had the inequality $w'(\vec{OPT}) \leq 2w(OPT)$ before, which means the weight of OPT may inflate when building the directed version $\vec{OPT}$, since an undirected edge is replaced by two directed edges. But now, this inflation is diminished by a factor of γ, since the weight of both directed copies is 2γ times the weight of the original undirected edge. We get

$$w'(\vec{OPT}) \leq 2\gamma w(OPT)$$

Note that this inequality will loose its validity for $\gamma < 1/2$, since then $2\gamma < 1$, but the weight of OPT need not deflate when changing to $\vec{OPT}$, for example if OPT contains only directed edges.
Combining the two inequalities, we get

$$w(A) \leq \frac{1}{\gamma} w'(I \cup O) \leq \frac{1}{\gamma}(w'(I) + w'(O)) \leq \frac{1}{\gamma} 2w(\vec{OPT}) \leq \frac{1}{\gamma} 4\gamma w(OPT)$$

Which again yields a factor of 4. □

From now on, we will always assume that the adapted weight function uses $\gamma = 1/2$. It is worth to look more closely on how w' and w change when switching between directed edge sets and mixed sets.

If $F \in E$ is a mixed edge set, then we have $w'(\vec{F}) = w(F)$, since an undirected edge gets replaced by two directed copies of half the weight each.

If $F \in \vec{E}$ is a directed edge set, we have $w(\bar{F}) = w(F) - w(F_d^U)/2$, since for a double undirected edge in F, only one instance is counted in $\bar{F}$. For w', we have the inequality $w(\bar{F}) = w'(F^D) + w'(F_d^U) + 2w'(F_s^U)$.

The above bound of 4 is tight, and in Figure 4.1 we give a tight example for $k = 2$. It is almost identical to the example in Figure 3.3, though all edges are undirected here. On this graph, the choice of γ obviously does not matter, because it multiplies all edge weights by the same factor.

The optimal 2-ECSS is of course the undirected Hamiltonian cycle $\{r, u\}, \{u, v\}, \{v, r\}$ and has weight $1 + \epsilon$. Our algorithm finds an optimal I containing the two edges $\{u, r\}_0, \{v, r\}_0$ having weight 0, and $\{u, r\}_1, \{v, r\}_1$, having weight 1 each. Now, we can assume that the algorithm unluckily includes into O the edges $\{r, u\}_2, \{r, v\}_2$, which have not yet been included in I and have a weight of 1 each, causing a ratio of 4. This seems pessimistic, since the algorithm might as well include $\{r, u\}_1, \{r, v\}_1$

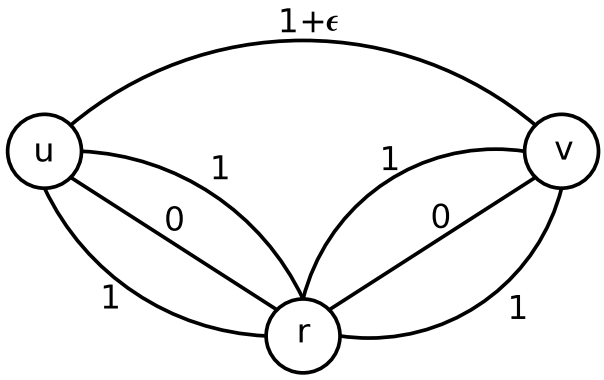

Figure 4.1: Tight example for the factor-4 approximation algorithm on mixed graphs for $k = 2$, using adapted weight function w'

into O, which are already in I. But in the worst case the algorithm makes the unlucky choice.

This example is worth some remarks: The branching algorithm without adapted weight function performs worst on graphs where the optimum uses many undirected edges in both directions. But there are tight examples where A is purely undirected as well as purely directed.

On the contrary, if we use the adapted weight function, the algorithm performs worst if A has mainly undirected edges and uses them in only one direction, whereas now OPT can be purely directed, undirected or anything in between. Thus, in an example that is tight for both adapted and nonadapted weight function, both A and OPT must contain mainly undirected edges.

The example in Figure 4.1 suggests that, after having constructed the k-in-tree I, we should give O an incentive to re-use edges in I. This might be carried out in different ways:

1. After constructing I using the adapted weight function w', we set the weight of every edge in I to 0. Here, if $e \in I^U$, we set both directions of e to 0. To be more precise, write $I^* := \vec{I}$, and set

$$w''(e) = \begin{cases} 0 & e \in I^* \\ w'(e) & e \notin I^* \end{cases}$$

Then we construct O for weight function w'', so O is a minimum weight k-out-tree **with respect to** w'', and return $A = \overline{I \cup O}$. We refer to this algorithm as the **incentive algorithm**.

Of course, lowering edge weights in I^* cannot increase the cost of the optimal k-out-tree. Writing O' for the minimum weight k-out-tree with respect to w', we have $w''(O) \leq w'(O)$, but $w'(O)$ can be

higher than $w'(O')$. Still, we have

$$w'(I \cup O) = w'(I) + w'(O \setminus I) \leq w'(I) + w''(O) \leq w'(I) + w'(O)$$

This simply says that edges in I might or might not be useful for O. But we cannot do worse than without setting the edges in I to 0.

2. As an alternative to the incentive algorithm, we can construct I and O as usual, i. e. using the adapted weight function w', and then explicitly try to use undirected edges in I to replace edges in O. We refer to this as the **simple replacement algorithm**, and state it in a more precise way later.

The simple replacement algorithm seems inferior to incentive algorithm, since it takes advantage of "reusable" edges only in a local way, whereas the latter does a global optimization. This is true, as we will see, but the simple replacement algorithm can more easily be analyzed and achieves a factor-3 approximation for a special case of our problem.

4.1 The Replacement Algorithm

Theorem 4.1 Given a mixed graph $G = (V, E)$ in which all undirected edges have uniform weights, and each directed edge has at least half the weight of an undirected edge. Then the simple replacement algorithm achieves an approximation ratio of 3.

Proof. The proof is very simple in principle and a little bit more involved in detail. We will show that an undirected edge in $I \setminus O$ can be used to replace an edge in O. This replacement does not necessarily reduce $w(A)$, since the edge that was replaced might have been also in I, thus leaving total weight unchanged. But we will show that after iteratively using edges in $I^U \setminus O$ to replace edges in O, we end up with sets I and O where $I^U \subseteq O$, and $w'(O)$ has not increased. Once we have achieved this, we have

$$\begin{aligned} & w(A) \\ \leq\ & w(I^D) + w(O^D) + w(I^U \setminus O^U) + w(O^U) \\ =\ & w(I^D) + w(O^D) + 0 + w(O^U) \\ =\ & w(I^D) + w(O) \leq w'(I) + 2w'(O) \\ \leq\ & 3w'(OPT) \leq 3w(OPT) \end{aligned}$$

So it remains to show how to transform I and O such that $I^U \subseteq O$.

Lemma 4.2 Given a mixed out-tree (in-tree) T rooted at r and an acyclic set of undirected edges F. Let $l = |F \setminus T|$. There is a set $J \subseteq T \setminus F$ with $|J| = l$ such that $T \cup F \setminus J$ is still a valid mixed out-tree (in-tree). We say we **swap** F into T.

Proof. We assume that T is an out-tree. The proof for in-trees is symmetric. First, we will prove the Lemma for connected F, i. e. when F is an undirected tree.

Let $u \in V(F)$ be a vertex that is maximal with respect to the tree T, i. e. no other $v \in V(F)$ is an ancestor of u in T. Taking u as the root of F automatically gives F an orientation. Now consider some edge $\{x, y\}$ in F, y being farther away from u in F than x. In T, y has a unique parent edge, $f = p_T(y)$, and we say f is the parent edge of $\{x, y\}$.

Define $p_T(F) := \{p_T(e), e \in F\}$. Of course, $|p_T(F)| = |F|$, since no two parent edges can be identical. Also note that $T \cap F \subseteq p_T(F)$. To see this, let $e = \{x, y\} \in T \cap F$ and assume w. l. o. g. that x is closer to u in F than y. If e has the same orientation in T as in F, i. e. x is closer to r in T than y, then e is its own parent edge and we are done.

Otherwise e is oriented (y, x) in T. Since F is a tree rooted at u, let e' be the parent edge of x in F. Then e is the parent edge of e'. e' always exists since x cannot be identical to u: u was chosen to be maximal in $V(F)$ with respect to T, but in T, y is an ancestor of x, so x is not maximal.

Now we define $J = p_T(F) \setminus F$ and claim that it has the properties stated in the Lemma. Since $|F| = |p_T(F)|$, we have $|F \setminus |p_T(F)| = |p_T(F) \setminus F|$. Thus, $|J| = |F \setminus p_T(F)| = |F \setminus T| = l$. Also, we must show that $T' := T \cup F \setminus J$ is 1-entering. For the sake of contradiction, suppose there is a set $X \subseteq V - r$ with $\rho_{T'}(X) = 0$. We know that $\rho_T(X) \geq 1$, since T is 1-entering. So there must be some parent edge $f = (x, y) \in J$ entering X. Hence, $y \in X \cap V(F)$. If $u \in X$, then the T-path from r to u enters X and does not contain any edges in J. If $u \notin X$, then F enters X. Both is a contradiction to the assumption that $\rho_{T'}(X) = 0$.

It remains to show that T' is a tree. But the number of edges that were removed, $|J|$, is equal to the number of edges that were added, $|F \setminus T|$. So T' must be a tree.

Now consider the case that F is a forest consisting of several trees $F_1, \ldots, F_s$. For each F_i, we have proven the existence of a set J_i. Note that the sets $V(F_i)$ are pairwise disjoint. This implies that the J_i are also pairwise disjoint, and so are the sets heads(J_i), the sets of all **heads** of the directed edges in the J_i.

This implies that the set J_i is still as required in the Lemma, even after having swapped $F_1, \ldots, F_{i-1}$ into T. Thus, $J := \bigcup_{i=1}^{s} J_i$ is as required in the Lemma. □

There are two points to remark: First, Lemma 4.2 does not hold any more when F contains directed edges, since we used the fact that edges in F can be oriented in any direction.

Second, in our special case, a swap does not increase $w'(T)$. This is because F contains only undirected edges. They have uniform weights, and for any directed edge f and undirected e, we have assumed $w(f) \geq w(e)/2$, hence $w'(f) \geq w'(e)$. It follows that $w'(F \setminus T) \leq w'(J)$.

Now we are ready to state the replacement algorithm in full detail:

(1) Let $I = I_1 \cup \cdots \cup I_k$ and $O = O_1 \cup \cdots \cup O_k$ be the decompositions of I and O, respectively, in k edge disjoint in-trees (out-trees, resp.)

(2) `while` $I^U \setminus O$ not empty:

  (a) pick some $e \in I^U \setminus O$ and let $e \in I_i$

  (b) set $F = I_i^U \setminus (\bigcup_{j \neq i} O_j^U)$, i. e. F includes all undirected edges of I_i^U that are not contained in an out-tree other than O_i.

  (c) swap F into O_i

In Step 2b we swap some undirected edges of I_i^U into O_i. When doing this, we have to make sure that these edges are not contained in any other O_j, because O_j and O_i still have to be edge-disjoint after the swap.

As we stated before, $w'(O)$ does not increase. Also, when the algorithm halts we have $I^U \subseteq O$. It remains to show that the algorithm ends after a polynomial number of iterations. To show this, we define the potential function

$$\begin{aligned} \Phi(I, O) &:= \sum_{i=1}^{k} |S_i| \\ S_i &:= (I_i^U \cap O_i^U) - \bigcup_{j \neq i} O_j^U \end{aligned}$$

The set S_i is the set of all undirected edges in I that are contained also in O_i, but in no other O_j. We claim that Φ increases in every iteration of the algorithm:

(1) The edge e chosen in Step 2a gets swapped into O_i and increases $|S_i|$ by one

(2) An edge f that was in S_i before the swap will be included in set F, which is chosen in Step 2b, and therefore will still be in O_i after the swap and contribute to Φ. Only the orientation of f in O_i may be changed by the swap.

(3) Suppose edge f was in some S_l for some $l \neq i$. Then f will still be in S_l after the swap unless it is contained in $\bigcup_{j \neq l} O_j^U$ after the swap. But the only O_j changed by the swap is O_i, hence f is still in S_j unless f has been swapped into O_i. But this cannot happen, because F does not contain f (remember in step 2b, we explicitly excluded edges from other O_j, $j \neq i$, from F).

So Φ increases by at least 1 every iteration and is bounded by $\sum_{i=1}^{k} |I_i^U| \leq 2|I^U|$. This proves the Lemma.[1] □

We can indeed go one step further: after having transformed O such that $I^U \subseteq O^U$, we can repeat the procedure with the roles of O and I switched, i. e. swapping undirected edges in $O \setminus I$ into I. After that, we will also have $O^U \subseteq I^U$, hence $O^U = I^U$. We call this the **full** replacement algorithm. We get

$$\begin{aligned} & w(\bar{A}) \\ \leq\ & w(I^D) + w(O^D) + w(I^U \cup O^U) \\ =\ & w(I^D) + w(O^D) + w(I^U) \\ \leq\ & w'(I^D) + w'(O^D) + 2w'(I^U) \\ =\ & w'(I) + w'(O) \leq 2w'(OPT) \\ \leq\ & 2w(OPT) \end{aligned}$$

where we use the fact that $w(I^U) = w(O^U) = w(I^U \cup O^U)$. This proves the main result of this Section:

Theorem 4.2 The **full replacement algorithm** achieves a factor 2 approximation for the special case where undirected edges have uniform weights and directed edges have weights at least half of that of an undirected edge. In particular the approximation ratio holds when all edges have equal weight.

4.2 Tight Examples for the Replacement Algorithm

One might ask if the replacement algorithm can be extended to more general cases, e. g. dropping the requirement that undirected edges must have uniform weights. The obvious problem now is that

[1] the factor 2 appears because an undirected edge can be contained in up to two sets I_i, in each direction once

the swap procedure might replace very cheap edges in O by expensive edges in I, thus barely reducing the weight of the solution.

There is a tight example on which the replacement algorithm fails to achieve anything better than a ratio of 4. For the global structure see Figure 4.2. The example consists of a cycle of cells $C_1, \ldots, C_n$. The cell C_1 contains only the root vertex r, and two neighboring cells are connected by four undirected edges. Every cell looks like the graph given in Figure 4.3. The edges drawn as coming from above, i. e. e, f, g and h connect this cell to the previous cell, and the edges drawn as coming from below connect it to the next cell. These edges act as the edges e', f', g' and h' of the next cell.

Edges inside a cell have weight 0, edges between cells have weight 1, except those between C_n and C_1, which have weight $1 + \epsilon$. This is simply to prevent the algorithm from finding the optimal 2-ECSS, which is a Hamiltonian path containing 1 expensive edge per cell, thus having total weight $n + \epsilon$. The different arcs parallel to the edges in Figure 4.3 indicate which edges are used by which of the four sets I_1, I_2, O_1, O_2 and in which direction. The weight of $I \cup O$ is easily seen to be $4(n-1)$.

The replacement algorithm might now pick edge e as an edge in $I_1^U \setminus O$ and swap $I_1^U \setminus O_2^U$ into O_1. In every cell the same now happens: the edges e, d are swapped into O_1 and replace edge a. To be more precise, edge a is the parent edge[2] of e in O_1, so e can replace a. On the other hand, d is its own parent edge in O_1, so d does not replace any edge.

Note that the edges c and d cannot be swapped into O_1, since they are contained in O_2, hence explicitly excluded by the algorithm from the swapping set F.

Now, we see that e has replaced a in O_1, i. e. a weight-0 edge was replaced by a weight-1 edge. In a second step, f is picked by the algorithm, and $I_2^U \setminus O_1^U$ is swapped into O_2. Per cell, this means edge f replaces $c \in O_2$.

This does not reduce the cost of $I \cup O$. Swapping $O^U \setminus I^U$ into I does not replace any expensive edges in I, either. After that swap, we end up with $O^U = I^U$, and the algorithm halts. But there has been no weight reduction, and the weight of A is still $4(n-1)$.

One might ask whether the replacement algorithm acts too cautiously here: when swapping $I_1^U \setminus O_2^U$ into O_1, one might well include edge b into F, since O_1 would use it in the other direction as O_2, which would still leave O_1 and O_2 edge disjoint. If we thus allowed F to grow beyond b, we could replace the expensive edge g. On the example in Figure 4.3, this "refined" replacement algorithm would achieve an approximation ratio of 2.

[2] The notion of the **parent edge** of an edge with respect to some tree is defined in the proof of Lemma 4.2

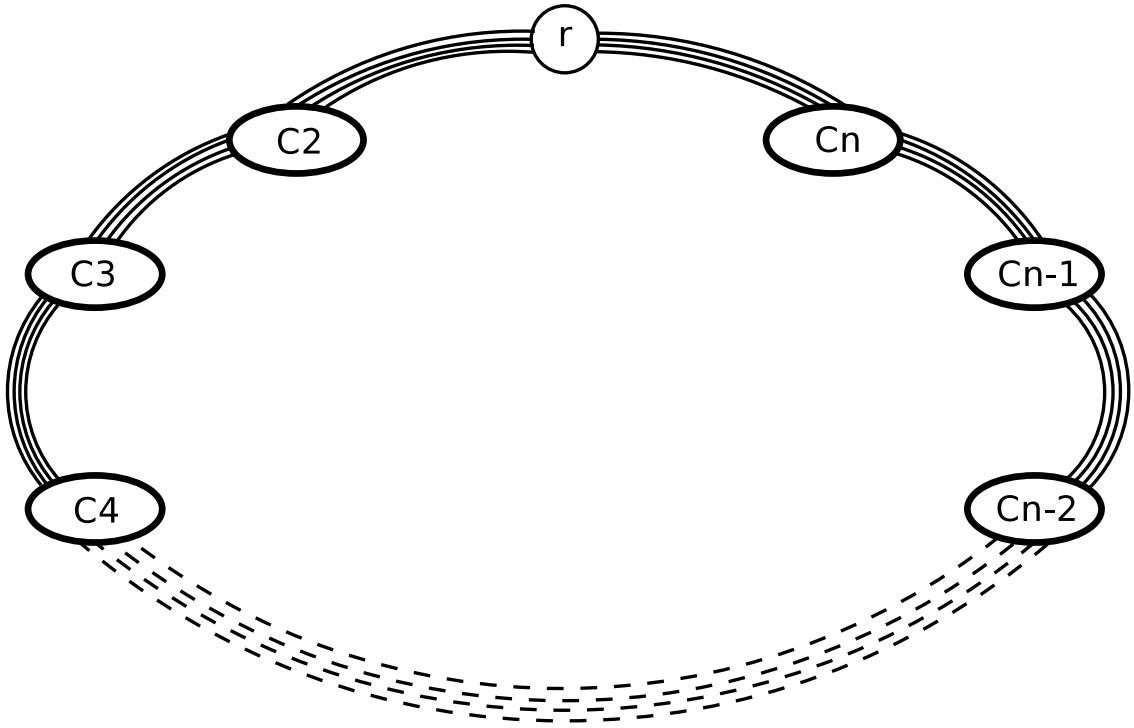

Figure 4.2: Example of poor performance of the replacement algorithm, for $k = 2$. Global structure of the example

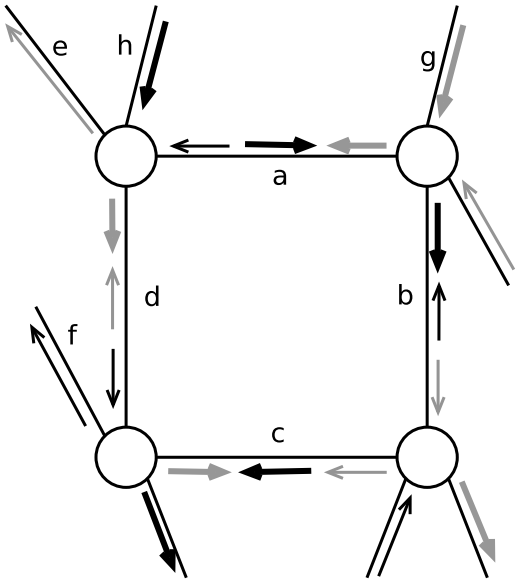

Figure 4.3: Local structure of the example. The small arcs indicate which of the four trees I_1, I_2, O_1, O_2 use which edges. Edges in I_1 are drawn grey/thin, in I_2 black/thin, in O_1 grey/bold and in O_2 black/bold

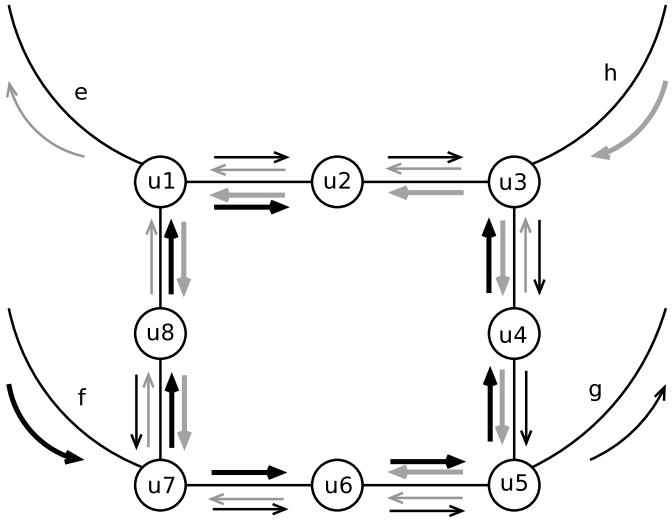

Figure 4.4: Example for poor performance of the refined replacement algorithm, for $k = 2$.

To state the refined algorithm more precisely, let $e = \{u, v\}$ be some edge in $I_i^U \setminus O$ that gets swapped into O_i. We assume w. l. o. g. that v is no ancestor of u in O_i. So e gets inserted into O_i in direction (u, v). From v, we can grow a search tree that contains edges in $I_i \cup O_i$ if they are not used by some other O_j in the same direction. However, we allow them if O_j uses them in the opposite direction.

Unfortunately, this algorithm does not yield an improvement either, and the tight example is only slightly more complicated. It is basically the same as for the normal replacement algorithm, only that we replaced the widget in Figure 4.3 by that in Figure 4.4.

Explanation of the example: Suppose we try to swap e into O_1. The search tree cannot grow from u_1 to u_2, since this edge is blocked by O_2. In can grow from u_1 to u_8 and u_7, but from u_7 it cannot grow in any direction. Even if we allowed the search tree to use the edge f in upwards direction, we would simply end up at the vertex corresponding to u_7 in the previous (above) cell.

These examples suggest that we should prefer the incentive algorithm, which would find a factor-2 approximation for both examples. But for now, let us examine another special case.

4.3 A Factor-3.75 Approximation Algorithm for a Special Case

In this section, we analyze the special case in which undirected edges are not more expensive than directed edges, but we do not require undirected edges to have uniform weights. Also, we restrict

our analysis to the case $k = 2$, since we suspect that a result for $k = 2$ should generalize to higher k with some additional work.

On a given graph, we will run the incentive algorithm as well as the normal branching algorithm with nonadapted weight function. We will show that the weight of the cheaper of the two solutions is at most 3.75 times the weight of an optimal 2-ECSS.

Definition 4.1 Let E be some edge set that is $k-1$-entering. F is called an augmenting set if $E \cup F$ is k-entering. Further, if each edge in F is nonredundant, i. e. for each edge $f \in F$, $E \cup (F - f)$ is not k-entering any more, we call F a **minimal augmenting set for E and k**. If there is no ambiguity about k, we simply call it a minimal augmenting set for E.

.

Definition 4.2 Let E be some edge set that is k-entering for some k. We call a set $M \subseteq V - r$ **critical with respect to** E if $\rho_E(M) = k$. If there is no ambiguity about E, we might simply call it critical.

If M and N are both critical w. r. t. some edge set E and $M \cap N$ is nonempty, then both $M \cap N$ and $M \cup N$ are critical as well. This follows from the submodularity of ρ:

$$2k = \rho_E(M) + \rho_E(N) \geq \rho_E(M \cup N) + \rho_E(M \cap N) \geq 2k$$

where the first inequality follows from submodularity, and the second from the fact that E is k-entering. Note that we have $r \notin M \cup N$, so $\rho_E(M \cup N) \geq k$. We get $\rho_E(M \cup N) = \rho_E(M \cap N) = k$.

Definition 4.3 Let F be a minimal augmenting set for E and k. Since F is minimal, $E \cup (F - f)$ is not k-entering any more. Therefore, there is some set M with $\rho_{E \cup (F-f)}(M) = k - 1$. This means, M is critical w. r. t. both E and $E \cup F$, and f is the only edge of F entering M. We call M a **critical set of** f.

Note that not every set that is critical w. r. t. E needs to be a critical set for an $f \in M$. Indeed, this is the case if $\rho_E(M) = k - 1$ and $\rho_F(M) \geq 2$.

Definition 4.4 Let $f \in F$, and $M_1, \ldots, M_n$ be all the critical sets of f. Then

$$M_f := \bigcap_{i=1}^{n} M_i$$

is nonempty, since every M_i contains the head of f. Further, M_f is critical w. r. t. both E and $E \cup F$, hence exactly one edge of F enters M_f, and this edge must be f. So M_f is also a critical set of f, and we call it the **minimal critical set** of f. From now on in this section, M_f will always denote the minimal critical set of f.

Define $\mathcal{M}_F := \{M_f, f \in F\}$, the family of all minimal critical sets of edges in F. Then every set $M \in \mathcal{M}_F$ is entered by exactly one edge of F and by exactly $k-1$ edges of E, and every $f \in F$ enters exactly one set in $\mathcal{M}_F$. Now we state in important fact about $\mathcal{M}_F$.

Proposition 4.1 $\mathcal{M}_F$ is a laminar family of sets, i. e. no two sets $M, N \in \mathcal{M}_F$ cross.

Proof. First recall that we say two sets M, N **cross** if none of the three sets $M \cap N$, $M \setminus N$ and $N \setminus M$ is empty. Sometimes, the definition of **cross** also requires $M \cup N \neq V$. But this trivially holds, since neither M nor N contains the root r.

Now, a family $\mathcal{M}$ of sets is called **laminar** if no two sets in $\mathcal{M}$ cross.

To prove the Proposition , suppose M_f and M_g intersect. Further, let $u = \text{head}(f)$ and $v = \text{head}(g)$. We observe that $N := M_f \cap M_g$ is critical w. r. t. both E and $E \cup F$.

If neither $v \in M_f$ nor $u \in M_g$, then N neither contains u nor v. Since $\rho_E(N) = k-1$, some edge $h \in F$ must enter it. This edge cannot be f or g, since N must contain the head of h. Also, h enters at least one of M_f and M_g, which is a contradiction, because both M_f and M_g are entered by exactly one edge of F.

So assume w. l. o. g. that $v \in M_f$. Then $v \in N$, so g enters N. Since $\rho_E(N) = k-1$ and $\rho_{E \cup F}(N) = k$, we have $\rho_F(N) = 1$, hence N is a critical set of g. But M_g is the **minimal** critical set entered by g, which implies $N = M_g$, hence $M_g \subseteq M_f$. □

Now we apply these results to our algorithm. Let A_{ad} be the solution returned by the incentive algorithm using adapted weight function w', and A_{na} the solution of the normal algorithm using nonadapted weight function w. We construct A_{ad} as before:

Construct the optimal 2-in-tree I, and let I^* denote $\vec{\vec{I}}$, i. e. if $(u, v) \in I$ comes from an undirected edge $\{u, v\} \in E$, we include also (v, u) into I^*. Note that this makes sense, since if I contains an undirected edge e, it will cause no additional cost if O uses it, no matter in which direction. Of course, I^* contains all the directed edges of I.

Then set $w''(e) = 0$ for every $e \in I^*$ and $w''(f) = w'(f)$ for every $f \notin I^*$. For w'', we construct the optimal 2-out-tree O and return $A_{ad} = \overline{I \cup O}$. Now we state the main result of this section.

Theorem 4.3 Let A_{ad} be the solution returned by the incentive algorithm using adapted weight function w' and A_{na} the solution of the branching algorithm using nonadapted weight function w. Let A be the cheaper solution. Then

$$\frac{w(A)}{w(OPT)} \leq 3.75$$

Proof. Decompose O into O_1 and O_2. Since O_1 and O_2 are edge disjoint and we have $w'(O) \leq w(OPT)$, we can assume w. l. o. g. $w'(O_1) \leq w(OPT)/2$.

Obviously, $O_1 \cup I^*$ is 1-entering. Hence, O_1 contains a minimal augmenting set S_1 that makes I^* 1-entering (this set can be empty).

Lemma 4.3 $|I^D| \geq 2|S_1|$, where $|I^D|$ is the number of directed edges in I.

Proof. We define $\mathcal{M}_F$ to be the family of all minimal critical sets of the edges $f \in S_1$. Therefore, every $M_f \in \mathcal{M}_F$ has $\rho_{I^*}(M_f) = 0$ and $\rho_{S_1}(M_f) = 1$.

Note that for $M_f \in \mathcal{M}_F$, no set $M_g \in \mathcal{M}_F$ that is a proper subset of M_f can contain the head of f, because otherwise f would enter M_g as well. So we see that the set

$$M_f^* := M_f \setminus \bigcup_{M_g \in \mathcal{M}_F, M_g \subset M_f} M_g$$

is nonempty for every f. Of course, the M_f^* are pairwise disjoint.

Since M_f^* is nonempty and I^* is 2-leaving, there are at least two edges $e_1, e_2 \in I^*$ leaving M_f^*. If one of them leaves M_f^* to $M_f \setminus M_f^*$, it enters some set $M_g \subset M_f$, which contradicts the fact that M_g is critical. So both edges leave M_f. If one of them was undirected, I^* would also include the edge in the other direction, which would enter M_f. This is again a contradiction, since $\rho_{I^*}(M_f) = 0$.

So for every M_f^*, I^* contains at least 2 directed edges leaving M_f^*. Since the M_f^* are pairwise disjoint, the directed edges leaving them are all distinct, which completes the proof. □

Since we assumed w. l. o. g. that $w'(O_1) \leq w(OPT)/2$, we also have $w(S_1^D) = w'(S_1^D) \leq w(OPT)/2$. Also, $|I^D| \geq 2|S_1|$, which implies $|I^D| \geq 2|S_1^U|$. Since we are analyzing the special case where undirected edges are no more expensive than directed edges, this translates to $w(S_1^U) \leq w(I^D)/2$. Hence, we derive

$$w(S_1) = w(S_1^D) + w(S_1^U) \leq w(OPT)/2 + w(I^D)/2$$

Now define $S_2 := O \setminus (I^* \cup S_1)$. Of course, S_2 is augmenting for $I^* \cup S_1$ and 2, i. e. it makes $I^* \cup S_1$ 2-entering. Also, S_2 is a **minimum weight** augmenting set for $I^* \cup S_1$ with respect to the

weight function w'. To see this, suppose for the sake of contradiction that F is a minimal augmenting set for $I^* \cup S_1$ and 2, and $w'(F) < w'(S_2)$. Then we have

$$w'(F \cup S_1) \leq w'(F) + w'(S_1) < w'(S_2) + w'(S_1) = w''(O)$$

The last equality follows from the fact that O might contain edges in addition to $S_2 \cup S_1$, but they have to be in I^*, so they do not add any weight to $w''(O)$. Also, since F and S_1 are minimal, they do not contain any edges from I^*, so $w''(F \cup S_1 \cup I^*) = w'(F \cup S_1)$. Of course, $F \cup S_1 \cup I^*$ is 2-entering. This is a contradiction, because O was constructed as a minimum weight 2-entering set with respect to w''. So S_2 is a minimum weight augmenting set. It need not be minimal, but if it is not, it contains some redundant edges that have weight 0. So we can remove them and assume that S_2 is also minimal.

As a result, every optimal solution must contain some minimal edge set F that makes $I^* \cup S_1$ 2-entering, and $w'(F) \geq w'(S_2)$. Therefore, we have $w(OPT) \geq w(F)$. Since F is a minimal augmenting set, we already know that $\mathcal{M}_F$, the family of all minimal critical sets of F, is laminar.

Proposition 4.2 Let l be the number of leaf sets of $\mathcal{M}_F$, i. e. the sets $M \in \mathcal{M}_F$ that are minimal in this family. Then $l \leq |I^D|$.

Proof. The proof is somewhat similar to that of Lemma 4.3. First note that all leaf sets are disjoint. Since I^* is 2-leaving, it contains at least two edges e_1, e_2 leaving each leaf $M \in \mathcal{M}_F$. If both of them are undirected, I^* contains also two edges entering M, hence $\rho_{I^*}(M) \geq 2$, which is a contradiction, because M is of course critical w. r. t. $I^* \cup S_1$. Hence every leaf is left by at least one directed edge of I, which proves the fact. □

We decompose F into the three sets F^D, F_d^U and F_s^U. F^D contains all edges of F that come from a directed edge in G, F_d^U contains an edge $e = (u, v)$ of F if it comes from an undirected edge and its twin edge, $e = (v, u)$, is also in F, i. e. if e is used in both directions. Finally, F_s^U contains all edges that come from an undirected edge, but are used in only one direction. We will establish the result $|F_d^U| \leq 2|I^D|$. Note that since we treat F as a directed edge set $\subseteq O\vec{P}T$, $|F_d^U|$ counts both two instances of an undirected edge used in both directions.

Lemma 4.4 Suppose F is a minimal augmenting set for $I^* \cup S_1$ and 2, then $|F_d^U| \leq 2|I^D|$.

Proof. First, enhance the laminar family $\mathcal{M}_F$ by the **root set** V. This set is of course neither critical, nor entered by any edge. The resulting $\mathcal{M}_F$ has a tree-like structure: for a non-root set M,

define the parent of M to be the minimal proper superset of M in $\mathcal{M}_F$. This parent is unique, because $\mathcal{M}_F$ is laminar.

Suppose F contains both $f = (u, v)$ and $f' = (v, u)$. Then we have $u \notin M_f$ and $v \notin M_{f'}$, so these sets must be disjoint (recall that $\mathcal{M}_F$ is laminar). We claim that M_f and $M_{f'}$ have the same parent. Let N be the parent of M_f. If it is the root set, we trivially have $M_{f'} \subset N$. If N is not a superset of $M_{f'}$, it is disjoint from $M_{f'}$, so f enters N as well. This cannot be, since f enters exactly one set of $\mathcal{M}_F$. So the parent of M_f is a superset of $M_{f'}$, and vice versa, hence their respective parents must be the same set.

Now, let $\mathcal{T}$ be the tree that represents the **parent**-relation in $\mathcal{M}_F$. Every pair $f = (u, v), f' = (v, u) \in F$ causes a pair of sets, $M_f, M_{f'}$ that have the same parent N. Call N the **parent** of the edge pair $\{f, f'\}$, and let pairs(N) denote the number of edge pairs whose parent is N. Now, the number of children of N in $\mathcal{T}$, short $d(N)$, is at least $2 * \text{pairs}(N)$. In the following calculation, N always denotes a non-leaf set, i. e. an interior node in $\mathcal{T}$. Let l be the number of leaves in $\mathcal{T}$, then we get

$$\begin{aligned} & l - 1 \\ = \; & \textstyle\sum_N (d(N) - 1) \\ \geq \; & \textstyle\sum_{N:\ \text{pairs(N)} \geq 1} (d(N) - 1) \\ \geq \; & \textstyle\sum_{N:\ \text{pairs(N)} \geq 1} (2 * \text{pairs(N)} - 1) \\ \geq \; & \textstyle\sum_{N:\ \text{pairs(N)} \geq 1} \text{pairs(N)} \\ = \; & |F_d^U|/2 \end{aligned}$$

This yields $|F_d^U| \leq 2l$, and Proposition 4.2 states that $l \leq |I^D|$. This proves the Lemma. □

We can derive the inequality $w'(F_d^U) \leq w(I^D)$. This follows from the cardinality inequality of Lemma 4.4 and the fact that under w', an edge that comes from an undirected edge weighs at most half the weight of a directed edge. Now, we can state an upper bound for $w'(S_2)$:

$$\begin{aligned} w(OPT) \geq w(F) = w'(F^D) + w'(F_d^U) + 2w'(F_s^U) = \\ 2w'(F) - w'(F^D) - w'(F_d^U) \Longrightarrow \\ 2w'(F) \leq w(OPT) + w'(F^D) + w(I^D) \end{aligned}$$

Since $w(S_2) \leq 2w'(S_2)$ and $w'(S_2) \leq w'(F)$ and $w(F^D) \leq w(OPT^D)$, this yields

$$w(S_2) \leq w(OPT) + w(OPT^D) + w(I^D) \tag{4.1}$$

For S_1, we already have an upper bound, which we state again here:

$$w(S_1) \leq \frac{1}{2} w(OPT) + \frac{1}{2} w(I^D) \tag{4.2}$$

Besides edges from S_1 and S_2, O contains only edges in I^* or other edges of weight 0. This follows because O is of minimum weight with respect to w''. Combining this fact with equation (4.1) and (4.2), we get

$$w(O \setminus I) \leq \frac{3}{2}w(OPT) + \frac{3}{2}w(I^D) + w(OPT^D) \tag{4.3}$$

Now we need to bound I. There are not many ways to do so, and the best we have is

$$w(I) \leq w(I^D) + 2w'(I^U) = 2w'(I) - w(I^D) \leq 2w(OPT) - w(I^D) \tag{4.4}$$

Adding 4.3 and 4.4, we get

$$w(A_{ad}) = w(I) + w(O \setminus I) \leq \frac{7}{2}w(OPT) + \frac{1}{2}w(I^D) + w(OPT^D) \tag{4.5}$$

Combining the trivial bound $w(O) \leq 2w(OPT)$ with (4.4), we get

$$w(A_{ad}) \leq 4w(OPT) - w(I^D) \tag{4.6}$$

Writing $w(I^D) = \alpha w(OPT)$ and $w(OPT^D) = \beta w(OPT)$ and $r_{ad} = w(A_{ad})/w(OPT)$, equations 4.5 and 4.6 yield the upper bound

$$r_{ad} \leq \min(\frac{7}{2} + \frac{\alpha}{2} + \beta, 4 - \alpha) \tag{4.7}$$

That is not enough to derive an approximation guarantee. But we know that $r_{na} := w(A_{na})/w(OPT)$, the approximation ratio of the branching algorithm using nonadapted weight function, is bounded by

$$\begin{aligned} & w(A_{na}) \\ \leq\ & w(I) + w(O) \\ \leq\ & 2w(O\vec{P}T) \\ =\ & 2(w(O\vec{P}T^D + w(O\vec{P}T^U) \\ =\ & 2w(OPT^D) + 4w(OPT^U) \\ =\ & 4w(OPT) - 2w(OPT^D) \end{aligned}$$

So r_{na}, the approximation ratio of the nonadapted algorithm, is

$$r_{na} \leq 4 - 2\beta \tag{4.8}$$

So if we run both algorithms, adapted and nonadapted, and return the cheaper solution, the ratio is

$$r = min(\frac{7}{2} + \frac{\alpha}{2} + \beta, 4 - \alpha, 4 - 2\beta) \tag{4.9}$$

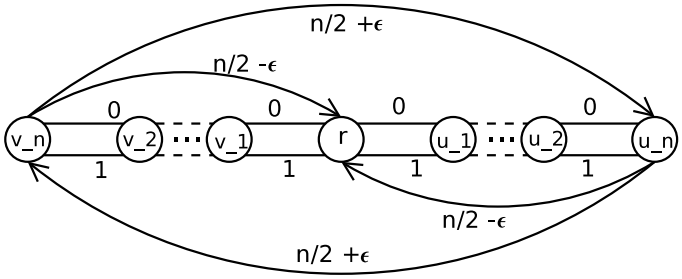

Figure 4.5: A factor-3 example for the incentive algorithm, $k = 2$, special case

It turns out that all three terms agree for $\alpha = 1/4$ and $\beta = 1/8$. This yields an approximation guarantee of $4 - 1/4$, or 3.75, which proves the Theorem. □

This result is not a big improvement over the factor 4 we have from the beginning, but it is a first step. I suppose the bound of 3.75 can further be tightened, especially since we did not even find an example on which the incentive algorithm alone performs worse than a factor of 3. We will give some examples in the next section.

4.4 Examples of Poor Performance of the Incentive Algorithm

In this section, we present two examples on which the incentive algorithm performs poorly. In the first example, directed edges are more expensive than undirected. Here, the incentive algorithm achieves an approximation ratio of 3. The second example is an instance of the general case, it contains cheap directed edges. Here, the algorithm finds a factor 4 approximation. Still, the algorithm using nonadapted weight function would find the optimal solution in this example.

For the factor-3 example see Figure 4.5. The optimal I is

$$\begin{aligned} I_1 &= \{(u_n, u_{n-1}), \ldots, (u_2, u_1), (u_1, r)\} \cup \{(v_n, v_{n-1}), \ldots, (v_2, v_1), (v_1, r)\} \\ I_2 &= \{(u_1, u_2), \ldots, (u_{n-1}, u_n), (u_n, r)\} \cup \{(v_1, v_2), \ldots, (v_{n-1}, v_n), (v_n, r)\} \end{aligned}$$

and has weight $w(I) = n - 2\epsilon$. Of course, I only includes the cheap (weight 0) edges between the u_i and v_i.

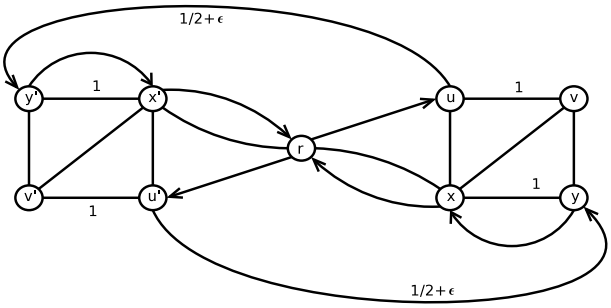

Figure 4.6: A factor-4 example for the incentive algorithm, $k = 2$, general case

Now, the optimal O with respect to w'' is

$$\begin{aligned} O_1 &= \{(u_n, u_{n-1}), \ldots, (u_2, u_1), (u_1, r)\} \cup \{(v_n, v_{n-1}), \ldots, (v_2, v_1), (v_1, r)\}, \\ &\quad \text{edges of weight 0} \\ O_2 &= \{(r, u_1), (u_1, u_2), \ldots, (u_{n-1}, u_n)\} \cup \{(r, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n)\}, \\ &\quad \text{edges of weight 1} \end{aligned}$$

This is optimal w. r. t. w'' only because w' assigns weight $1/2$ to the expensive undirected edges, making this option cheaper than using the two undirected edges (u_n, v_n) and (v_n, u_n). We have $w(O) = 2n$, which yields $w(A) = 3n - 2\epsilon$. The optimal solution is easily seen to consist of all weight-0 edges plus the two directed edges (u_n, v_n) and (v_n, u_n). This gives $w(OPT) = n + 2\epsilon$.

Thus, the incentive algorithm achieves a ratio of 3 on this example. That is the worst example we have found. Also note that the branching algorithm would achieve a factor 2 approximation here.

The second example is a factor 4 example for the incentive algorithm on the general case, i. e. we allow directed edges to be cheaper than undirected. The example is given in Figure 4.6. The unlabelled edges have weight 0. OPT, I and O are less obvious on this example than on the example before. I is given in Figure 4.7. The grey and black edges are those of I_1 and I_2, respectively, the dashed edges are those not in I. Similarly, see Figure 4.8 for the set O. The edges included by I either have weight 0 or are of no use for O. The same holds for the edges of O. Together, I and O have a weight of 4.

The optimal solution (See Figure 4.9) has a weight of $1 + 2\epsilon$, so the approximation ratio comes arbitrarily close to 4. Here, the branching algorithm with nonadapted weight function would find the optimal solution.

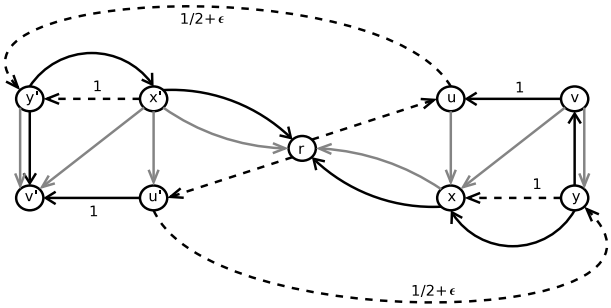

Figure 4.7: The set I of the factor-4 example

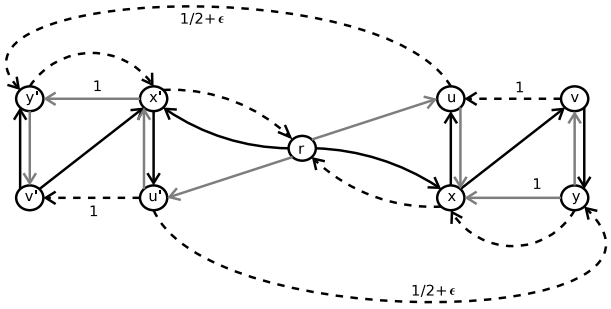

Figure 4.8: The set O of the factor-4 example

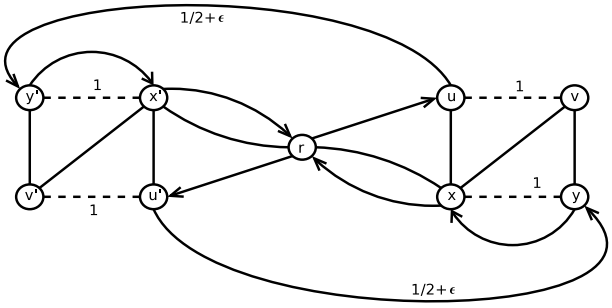

Figure 4.9: The optimal 2-ECSS of the factor-4 example

4.5 Cheap Directed Edges

In the last section, we have examined the special case that undirected edges are no more expensive than directed edges (or have at least half the weight of the latter). Now, we restrict ourselves to the case where every directed edge has weight 0 and every undirected edge has weight 1. Gabow [4] proved that Jain's iterated rounding algorithm [6] achieves a factor 2 approximation for the directed Steiner network problem where all directed edges have cost 0, which of course implies a factor 2 approximation for the problem in this section.

I have not found a proof that the incentive algorithm achieves anything better than a ratio of 4 in this special case, and did not find a tight example, either. However, I got some results that might be useful. Again, we restrict our attention to the case $k = 2$, since we suspect once a better approximation for $k = 2$ has been found, it will generalize to higher k.

As before, let I be a minimum weight 2-in-tree and O a minimum weight 2-out-tree, let $I = I_1 \cup I_2$ and $O = O_1 \cup O_2$ be the partition of I and O into 2 edge disjoint in-trees and out-trees, respectively.

The replacement algorithm uses the fact that after I and O have been constructed, several edges in $\overline{I \cup O}$ might be redundant. What pushes the approximation ratio above 2 is the possibility of A containing many undirected edges which are used only once in the four sets I_1, I_2, O_1 and O_2. At this point, we must change our previous notion of a single edge. From now on, we call an edge **single** if it is in only one set of I_1, I_2, O_1 and O_2. Note that it is possible that a non-single edge is used in only one direction, say, by I_1 and O_1, since these sets need not be disjoint.

Since all directed edges are assumed to have weight 0, we simply include all of them in A. Suppose now we remove redundant single undirected edges (edges of weight 1) in an arbitrary order, then we are left with a solution $A = I \cup O$ where every single undirected edge e in O (in I) enters (leaves) a set C_e with $\rho_A(C_e) = 2$ ($\delta_A(C_e) = 2$). We call C_e the **critical set** of edge e. Since the intersection of two critical sets is either empty or itself critical, every single undirected edge enters a unique minimal critical set. From now on, C_e will denote this minimal critical set which e enters (or leaves, respectively, if $e \in I$), and e will always denote a single undirected edge.

Also, we only write C_f for some edge f if f is a single undirected edge. In this section, e will always denote an edge in O_1. For all other cases the results and proofs are similar.

We now list several facts about the C_e.

Proposition 4.3 C_e does not contain the root r.

Proof. Suppose $r \in C_e$. Since $C_e \neq V$, $V \setminus C_e$ is non-empty and does not contain the root. Therefore ≥ 2 edges of I leave $V \setminus C_e$, hence enter C_e. Those edges cannot be identical to e, since e is a single edge. So $\rho_A(C_e) \geq 3$, which is a contradiction. □

Proposition 4.4 Besides e, there is exactly one edge of A entering C_e. This edge belongs to O_2 and is called enter(C_e).

Proof. O_2 enters C_e because $r \notin C_e$. □

As a consequence, O_1 enters C_e only once, therefore C_e is a subset of the subtree of O_1 rooted at the head of e.

Proposition 4.5 Let C_e be the minimal critical set of $e = (u, v)$ and enter$(C_e) = (x, y)$. Then there are two directed edge disjoint paths inside C_e from y to v.

Proof. This follows from the minimality of C_e. If C_e did not contain two such paths, there would be a (y-v)-cut $C_e = C_e^1 \cup C_e^2$ containing only one edge, so C_e would not be minimal[3] . □

A similar argument shows that for every vertex $x \in C_e$ there is a path inside C_e from x back to v. Also, since O_1 enters C_e exactly once, there is a path from v to x inside C_e, containing only O_1-edges. Further, only one directed edge of G can enter C_e. This is because e is an undirected edge entering C_e, and every further directed edge in G entering C_e increases $\rho_A(C_e)$ by one (because we included all directed edges into A). So if a directed edge enters C_e, it must be enter(C_e). This implies that every 2-ECSS, especially OPT, must contain at least one undirected edge adjacent to C_e.

Now we state the main result of this section, which says that the minimal critical sets are "almost" laminar.

Lemma 4.5 Let C_e and $C_{e'}$ be the minimal critical sets of two single undirected edges $e \in O_1$ and $e' \in A$. Then C_e and $C_{e'}$ can cross only if $e' \in O_2$, e is outside $C_{e'}$, e' outside C_e, and then $C_{e,e'} := C_e \cup C_{e'}$ is critical itself and does not cross with any C_f. In this case, we call C a heavy critical set. Note that $C_{e,e'}$ is not a minimal critical set any more.

[3] it might be that $y = v$. But in this case, we have the empty path, which is trivially edge disjoint with itself.

Of course, a similar result holds if e is in O_2, I_1 or I_2. Note also that C_e and $C_{e'}$ for $e \in O_1$ and $e' \in O_2$ can be identical. Then we also call $C_e = C_{e'}$ a heavy set.

If we unify all crossing C_e to heavy critical sets, the resulting family $\mathcal{C} = \{C_e, \text{ not heavy }\} \cup \{C_{e,f}, \text{heavy}\}$ is **weakly laminar** in the sense that a set can occur more than once (at most twice, though).

Also, for every set $C \in \mathcal{C}$, heavy or not, OPT contains at least as many undirected edges adjacent to C as A does. Unfortunately, I did not succeed deriving an approximation guarantee from this fact, since there is no upper bound on the number of sets C an undirected edge in OPT can leave.

To prove Lemma 4.5, we need to establish three facts:

(1) C_e cannot cross with $C_{e'}$ if $e' \in O_1$

(2) C_e cannot cross with $C_{e'}$ if $e' \in I$

(3) if $e' \in O_2$ and C_e, $C_{e'}$ cross, then only under the conditions described in the Lemma.

During the proof, we will always write $e' = \{u', v'\}$ and $e = \{u, v\}$.

Proposition 4.6 If e and e' are in O_1, then their critical sets cannot cross.

Proof. Since O_1 is a tree, the subtrees of v and v' cannot cross. If they are disjoint, then so are the critical sets and we are done. So we assume w. l. o. g. e' is in the subtree of e. If e' is outside C_e or leaving C_e, then the critical sets are disjoint, and again we are done. So we assume that e' is inside C_e.

Since both C_e and $C_{e'}$ are critical, their intersection, call it C, is also critical. Further, e' enters C. Since $C_{e'}$ is the minimal critical set e' enters, we get $C = C_{e'}$, hence $C_{e'} \subset C_e$. □

Proposition 4.7 If $e \in O_1$ and $e' \in I_1$, their critical sets cannot cross.

Proof. First note that C_e needs not be in-critical (there might be more than 2 edges leaving C_e, and $C_{e'}$ needs not be out-critical. So we cannot say anything about their intersection being critical or not.

e' cannot be adjacent to C_e, otherwise both e' and enter(C_e) would enter C_e (e' and enter(C_e) cannot be identical, otherwise e' would not be a single undirected edge). Similarly, e cannot be adjacent to $C_{e'}$. Note that an undirected edge adjacent to a set C always counts as entering C as well as

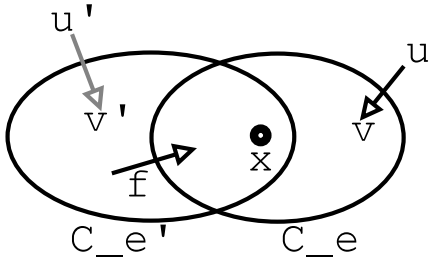

Figure 4.10: $e = (uv) \in O_1$ and $e' = (v'u') \in I_1$, with e' being outside C_e

leaving it.

Case 1: e is outside $C_{e'}$ and e' is outside C_e (See Figure 4.10. Edges in O are drawn black, edges in I grey). Pick some vertex $x \in C_e \cap C_{e'}$. There is a path from v' to x in $C_{e'}$. This path enters C_e via $f = \text{enter}(C_e)$. Let $f = (a, b)$. There are two edge disjoint paths in C_e from b to v. Since $b \in C_{e'}$ and $v \notin C_{e'}$, both are leaving $C_{e'}$, and neither of them can be e', implying $\delta_A(C_{e'}) \geq 3$, which is a contradiction.

Case 2: See figure 4.11. We can assume w. l. o. g. that e' is inside C_e, the other case is similar.

Now pick some vertex $x \in C_{e'} \setminus C_e$. Since I_1 is an in-tree, there is an I_1-path from x to v' in $C_{e'}$. This path enters C_e via some edge f. This edge is in I_1, hence cannot be e, so this edge must be enter(C_e). Now consider $f' = \text{leave}(I_2)$, the unique edge of I_2 leaving $C_{e'}$. Say $f' = (a, b)$ and suppose $b \in C_e$. Then the I_2-path from x to r must leave $C_{e'}$ at some point via f'. Hence it enters C_e via some edge that cannot be identical to f, since f is already in I_1, and I_1, I_2 are disjoint. It cannot be e, either, since e is single. This again implies $\rho_A(C_e) \geq 3$, which is a contradiction.

So suppose $b \notin C_e$. Then the O_2-path from r to some $y \in C_e \setminus C_{e'}$ enters C_e via f, which is inside $C_{e'}$. Hence the O_2-path must leave $C_{e'}$ again on its way to y. Since this path stays in C_e once it has entered it, it cannot leave $C_{e'}$ via f', but neither via e', since e' is single. This implies $\delta_A(T_2) \geq 3$, leading again to a contradiction. □

Proposition 4.8 If $e \in O_1$ and $e' \in O_2$, then they can only cross if e' is outside C_e and e is outside $C_{e'}$, and then $C_{e,e'} := C_e \cup C_{e'}$ is a heavy set, i. e. it is entered by no edges of A besides e and e'.

Proof. Again, we distinguish two cases. In Case 1, we assume that e' is inside C_e or entering

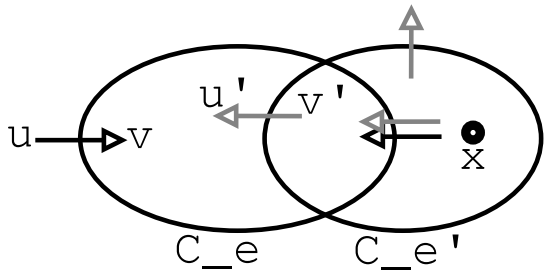

Figure 4.11: $e = (uv) \in O_1$ and $e' = (v'u') \in I_1$, with e' being inside C_e

it, i. e. $v' \in C - e$. Note that e' cannot leave C_e, since O_2 also enters C_e, implying $\rho_A(C_e) \geq 3$, (even if e' leaves C_e, it is also counted as entering C_e, since it is undirected). In Case 2, we assume $u, v \notin C_{e'}$ and $u', v' \notin C_e$.

Case 1: $v' \in C_e$ (the case $v \in C_{e'}$ is similar). Then $C := C_e \cap C_{e'}$ is critical, i. e. $\delta(C) = 2$. Since $v' \in C_e$ as well, e' enters C. Since $C_{e'}$ was chosen to be the minimal set that e' enters, we get $C = C_{e'}$, hence $C_{e'} \subseteq C_e$. Indeed, we can even have $C_{e'} = C_e$.

Case 2: e is outside $C_{e'}$, and e' is outside C_e

Since the intersection of the two sets is nonempty, their union $C_{e,e'} := C_e \cup C_{e'}$ is also critical, i. e. $\delta(C_{e,e'}) = 2$. The two edges entering $C_{e,e'}$ are of course e and e' themselves. So there are no edges in A besides those two entering $C_{e,e'}$. Also, no directed edge of G, hence none of OPT, either, enters it.

The heavy set $C_{e,e'}$ cannot cross with any C_f, $f \in O$, since this would increase the number of edges entering it. Neither can it cross with C_f, $f \in I$, since then either C_e or $C_{e'}$ would cross with C_f, which has already been shown to be impossible.

This proves the Lemma. □

Chapter 5

Summary

The incentive algorithm seems to be a natural approach to the k-ECSS problem on mixed graphs. Unfortunately, both the proof of the factor 2 approximation in Section 4.1 and the proof of the factor 3.75 in Section 4.3 contain a counting argument, therefore they have to make assumptions about the weights of undirected edges, which makes it difficult to generalize to arbitrary edge weights.

As stated before, the ratio of 3.75 seems far from being tight, and I neither managed to find an example on which the incentive algorithm as well as the branching algorithm achieve an approximation ratio strictly larger than 2. For the incentive algorithm alone, we gave a factor 3 example for the special case of cheap undirected edges and a factor 4 example for the general case.

We conclude this thesis with the conjecture that the combined incentive / branching algorithm achieves a ratio of less than 4 in the general case, and that the incentive algorithm alone performs better than 4 on the special case of cheap undirected edges.

Bibliography

[1] J. Edmonds. Edge-disjoint branchings. In Combinatorial Algorithms (R. Rustin, Ed.), Algorithmic Press, New York, pages 91–96, 1972.

[2] A. Frank. Kernel systems of directed graphs. In Acta Sci. Math. 41, pages 63–76, 1979.

[3] H. N. Gabow. Better performance bounds for finding the smallest k-edge connected spanning subgraph of a multigraph. In SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms, pages 460–469. Society for Industrial and Applied Mathematics, 2003.

[4] H. N. Gabow. On the L_∞-norm of extreme points for crossing supermodular directed network LPs. In 11th MPS Conference on Integer Programming and Combinatorial Optimization (IPCO), to appear 2005.

[5] H. N. Gabow, M. X. Goemans, E. Tardos, and D. P. Williamson. Approximating the smallest k-edge connected spanning subgraph by LP-rounding. In SODA '05: Proceedings of the 16th annual ACM-SIAM symposium on Discrete algorithms, pages 562–571, 2005.

[6] K. Jain. A factor 2 approximation algorithm for the generalized steiner network problem. In Combinatorica 21, pages 39–60, 2001.

[7] S. Khuller and B. Raghavachari. Improved approximation algorithms for uniform connectivity problems. In J. Algorithms 21, 2, pages 434–450, 1996.

[8] E. Lawler. Matroid intersection algorithms. In Math. Programming 9, pages 31–56, 1975.