Egerváry Research Group
on Combinatorial Optimization

Technical reportS

TR-2014-02. Published by the Egerváry Research Group, Pázmány P. sétány 1/C, H–1117, Budapest, Hungary. Web site: www.cs.elte.hu/egres . ISSN 1587–4451.

Blocking unions of arborescences

Attila Bernáth and Gyula Pap

April 2014

Blocking unions of arborescences

Attila Bernáth* and Gyula Pap**

Abstract

Given a digraph $D = (V, A)$ and a positive integer k, a subset $B \subseteq A$ is called a **k-union-arborescence**, if it is the disjoint union of k spanning arborescences. When also arc-costs $c : A \to \mathbb{R}_+$ are given, minimizing the cost k-union-arborescence is well-known to be tractable. In this paper we take on the following problem: what is the minimum cardinality of a set of arcs the removal of if which destroys every minimum c-cost k-union-arborescence. Actually, the more general weighted problem is also considered, that is, arc weights $w : A \to \mathbb{R}_+$ (unrelated to c) are also given, and the goal is to find a minimum weight set of arcs the removal of if which destroys every minimum c-cost k-union-arborescence. An equivalent version of this problem is where the roots of the arborescences are fixed in advance. In an earlier paper [2] we solved this problem for $k = 1$. This work reports on another partial result on the problem: we solve the case when $c \equiv 1$ is uniform – that is, the set of arcs needs to cover all k-union-arbosercences. The solution uses a result of Bárász, Becker and Frank [1] saying that the family of so-called insolid sets (sets with the property that every proper subset has a larger in-degree) satisfies the Helly-property, and thus can be (efficiently) represented as a subtree hypergraph.

Keywords: arborescences, polynomial algorithm, covering

1 Introduction

Let $D = (V, A)$ be a digraph with vertex set V and arc set A. A **spanning arborescence** is a subset $B \subseteq A$ that is a spanning tree in the undirected sense, and every node has in-degree at most one. Thus there is exactly one node, the **root node**, with in-degree zero. Equivalently, a spanning arborescence is a subset $B \subseteq A$ with the

*SZTAKI, Institute for Computer Science and Control, Lágymányosi u. 11, Budapest, Hungary H-1111. Part of the research was done while the author was at Warsaw University, Institute of Informatics, ul. Banacha 2, 02-097 Warsaw, Poland. Research was supported by the ERC StG project PAAl no. 259515. E-mail: bernath@cs.elte.hu.

**MTA-ELTE Egerváry Research Group, Department of Operations Research, Eötvös University, Pázmány Péter sétány 1/C, Budapest, Hungary, H-1117. Supported by OTKA grant no. K109240. E-mail: gyuszko@cs.elte.hu.

property that there is a root node $r \in V$ such that $\varrho_B(r) = 0$, and $\varrho_B(v) = 1$ for $v \in V - r$, and B contains no cycle. We will also call a spanning arborescence as a **arborescence** for short, when the set of nodes is obvious from context. If $r \in V$ is the root of spanning arborescence B then B is said to be an r-**arborescence**.

Given also a positive integer k, a subset $B \subseteq A$ is called a k-**union-**r-**arborescence**, if it is the arc-disjoint union of k spanning arborescences. In the special case when every arborescence has the same root r we call B a k-**union-**r-**arborescence**.

Given $D = (V, A)$, k and a cost function $c : A \to \mathbb{R}_+$, it is well known to find a **minimum cost** k-**union-arborescence** in polynomial time, and to find a **minimum cost** k-**union-**r-**arborescence** just as well. See [8], Chapter 53.8 for a reference, where several related problems are considered. The existence of a k-union-r-arborescence is characterized by Edmonds' disjoint arborescence theorem, while the existence of a k-union-arborescence is characterized by a theorem of Frank [4]. Frank also gave a linear programming description of the convex hull of k-union-arborescence, generalizing Edmonds' linear programming description of the convex hull of k-union-r-arborescences. The problem of finding a minimum cost k-union-r-arborescence may also be solved with the use of these results, either via a reduction to minimum weight k-union-r-arborescences, or minimum weight matroid intersection.

In this paper we consider the following covering problems, which are polynomial time equivalent.

Problem 1 (**Blocking optimal** k-**union-arborescences**)**.** Given a digraph $D = (V, A)$, a positive integer k, a cost function $c : A \to \mathbb{R}_+$ and a nonnegative weight function $w : A \to \mathbb{R}_+$, find a subset H of the arc set such that H intersects every minimum c-cost k-union-arborescence, and $w(H)$ is minimum.

Here the expression "intersects" simply means that the two have nonempty intersection. We remark that Problem 1 is polynomially equivalent with the version where the root is also given in advance, that is, the problem of blocking optimal k-union-r-arborescences.

Problem 2 (**Blocking optimal** k-**union-**r-**arborescences**)**.** Given a digraph $D = (V, A)$, a node $r \in V$, a positive integer k, a cost function $c : A \to \mathbb{R}_+$ and a nonnegative weight function $w : A \to \mathbb{R}_+$, find a subset H of the arc set such that H intersects every minimum c-cost k-union-arborescence, and $w(H)$ is minimum.

In section 3 we will show that the two problems are polynomial time equivalent.

In our previous paper [2] we have solved the special case of Problem 1 when we fix $k = 1$. Our conjecture is that the problem is also polynomial time solvable when k is not fixed. The main result of this paper is that the problem is polynomial time solvable when k is part of the input, and c is set to be constant – that is, when H is required to intersect *every* k-union-arborescence.

We remark that the version of Problem 1 and Problem 2 where we set c to be constant are *not* easily seen to be equivalent - actually, Problem 2 with c constant is solvable as a corollary of Edmonds' disjoint arborescences theorem by taking all but $k-1$ arcs from a minimum cut. Thus it is important to note that in the sequel we will consider the version of Problem 1 (and not Problem 2 !) with constant cost function c.

2 Notation

Let us overview some of the notation and definitions used in the paper. The arc set of the digraph D will also be denoted by $A(D)$. Given a digraph $D = (V, A)$ and a node set $Z \subseteq V$, let $D[Z]$ be the digraph obtained from D by deleting the nodes of $V - Z$ (and all the arcs incident with them). If $B \subseteq A$ is a subset of the arc set, then we will identify B and the subgraph (V, B). Thus $B[Z]$ is obtained from (V, B) by deleting the nodes of $V - Z$ (and the arcs of B incident with them). The set of arcs of D entering Z is denoted $\delta^{in}_D(Z)$, the number of these arcs is $\varrho_D(Z) = |\delta^{in}_D(Z)|$. An **arc-weighted digraph** is a triple $D_w = (V, A, w)$ where (V, A) is a digraph and a weight function $w : A \to \mathbb{R}_+$. For an arc-weighted digraph $D_w = (V, A, w)$ and subset X of its node set we let $\varrho_{D_w}(X) = \sum\{w_a : a \text{ enters } X\}$ denote the weighted indegree. A **subpartition** of a subset X of V is a collection of pairwise disjoint non-empty subsets of X: note that $\emptyset$ cannot be a member of a subpartition, but $\emptyset$ is a valid subpartition, having no members at all. For a vector $x : A \to \mathbb{R}$ and subset $Z \subseteq A$ we let $x(Z) = \sum_{v \in Z} x_v$.

3 Equivalence of versions

Theorem 3.1. *The following problems are polynomially equivalent (where $D_i = (V_i, A_i)$ is a digraph, k a positive integer, $c_i : A_i \to \mathbb{R}_+$ and $w_i : A_i \to \mathbb{R}_+$ for $i = 1, 2$).*

1. ***Blocking optimal k-union-arborescences (Problem 1):*** *Given D_1, k, c_1, w_1, find $H \subseteq A_1$ so that H intersects every minimum c_1-cost k-union-arborescence in D_1, and $w_1(H)$ is minimum.*

2. ***Blocking optimal k-union-r-arborescences (Problem 2):*** *Given D_2, k, c_2, w_2 and a node $r \in V_2$, find $H \subseteq A_2$ so that H intersects every minimum c_2-cost k-union-r-arborescence in D_2, and $w_2(H)$ is minimum.*

Proof. Problem 2 reduces to Probem 1 by deleting all arcs entering node r from the input digraph. For the reverse reduction, consider an instance D_1, k, c_1, w_1 of the Problem 1, and define an instance of Problem 2 as follows. Let $V_2 = V_1 + r$ with a new node r, and let $D_2 = (V_2, A_1 \cup \{rv : v \in V_1\})$. Let the costs defined as $c_2(a) = c_1(a)$ for every $a \in A_1$ and $c_2(rv) = C$ for every new arc $rv (v \in V_1)$ where $C = \sum_{a \in A_1} c_1(a) + 1$. Finally, the weights are defined as follows: $w_2(a) = w_1(a)$ for every $a \in A_1$ and $w_2(rv) = W$ for every new arc $rv (v \in V_1)$ where $W = \sum_{a \in A_1} w_1(a) + 1$. By these choices, in the instance to Problem 2 given by D_2, k, c_2, w_2 and r, the minimum c_2-cost k-union-r-arborescences naturally correspond to minimum c_1-cost k-arborescences in D_1 (since they will use exactly one arc leaving r), and for blocking these we will not use the new arcs because of their large weight. □

4 Solution of the uniform cost case

In this section we solve Problem 1 in the case when c is uniform. We first recall some definitions and results to be used later. The following result is fundamental for proofs in this paper. A **subpartition** of V is a collection of disjoint nonempty subsets of V.

Theorem 4.1 (Frank, [5, 4]). *Given a digraph $D = (V, A)$ and a positive integer k, there exists a k-union-arborescence in D if and only if $\sum_{X\in\mathcal{X}} \varrho_D(X) \geq k(|\mathcal{X}| - 1)$ for every subpartition $\mathcal{X}$ of V.*

Note that the condition in the theorem above always (trivially) holds for subpartitions having only one member, thus we may only need to check for subpartitions having at least two members. Furthermore, we may also narrow down to subpartitions in which every member is an insolid set of the digraph - the notion of insolid sets is defined as follows.

Definition 1. Given a digraph $G = (V, A)$, a non-empty subset of nodes $X \subseteq V$ is called *in-solid*, if $\varrho(Y) > \varrho(X)$ holds for every nonempty $Y \subsetneq X$.

A simple but useful corollary of the definition is the following fact.

Claim 4.2. *Given a digraph $D = (V, A)$ and an arbitrary non-empty subset $X \subseteq V$, there exists an in-solid set $X' \subseteq X$ such that $\varrho_D(X') \leq \varrho_D(X)$.*

This implies that Frank's theorem can be formulated with insolid sets.

Theorem 4.3 (Equivalent form of Theorem 4.1). *Given a digraph $D = (V, A)$ and a positive integer k, there exists a k-union-arborescence in D if and only if $\sum_{X\in\mathcal{X}} \varrho_D(X) \geq k(|\mathcal{X}| - 1)$ for every collection $\mathcal{X}$ of disjoint insolid sets of V. (Such a family $\mathcal{X}$ is called an insolid subpartition.)*

By Theorem 4.1, our Problem 1 reduces to finding a smallest cardinality subset of arcs which, when removed, will create a violating subpartition. A subpartition $\mathcal{X}$ becomes a violating subpartition if we remove at least $\sum_{X\in\mathcal{X}} \varrho_D(X) - k(|\mathcal{X}|-1)+1$ arcs from $\cup\{\delta^{in}(X) : X \in \mathcal{X}\}$. Note that this is only possible if $\sum_{X\in\mathcal{X}} \varrho_D(X) - k(|\mathcal{X}|-1)+1 \leq \sum_{X\in\mathcal{X}} \varrho_D(X)$, which is equivalent to $|\mathcal{X}| \geq 2$. Furthermore, by Theorem 4.3, we can narrow down to subpartitions consisting of insolid sets (**insolid subpartitions**). Therefore Problem 1 with uniform cost c is equivalent to the following problem.

Problem 3. Given a digraph $D = (V, A)$ and a positive integer k, find an insolid subpartition $\mathcal{X}$with $|\mathcal{X}| \geq 2$ maximizing $\sum_{X\in\mathcal{X}}(k - \varrho_D(X))$.

We solve this problem in two steps: first we show how to solve it without the requirement on the size of the subpartition, and then we show how to force that requirement. Note that a subpartition $\mathcal{X}$ maximizing $\sum_{X\in\mathcal{X}}(k - \varrho_D(X))$ does not automatically satisfy the requirement $|\mathcal{X}| \geq 2$: for example in a k-arc-connected digraph $\mathcal{X} = \{V\}$ is an optimal subpartition. Note that the problem without the size requirement is a maximum weight matching problem in the hypergraph of insolid sets, but with the special weight function $k - \varrho$.

4.1 Finding an optimal subpartition of unconstrained size

In this section we solve the variant of Problem 3 that comes without the requirement on the size of the subpartition. In fact, we will need to solve a little more general problem, namely the following one.

Problem 4. Given a weighted digraph $D_w = (V, A, w)$, a nonempty subset $V' \subseteq V$ and a positive integer k, find a subpartition $\mathcal{X}$ of V' maximizing $\sum_{X\in\mathcal{X}}(k - \varrho_{D_w}(X))$.

Note that we could restrict ourselves to insolid subpartitions in the problem. The following simple observation will be useful later.

Claim 4.4. *Given an instance to Problem 4, the subpartition $\mathcal{X} = \emptyset$ is an optimal solution if and only if $\varrho(X) \geq k$ for every nonempty $X \subseteq V'$.*

For an arbitrary arc-weighted digraph $D_w = (V, A, w)$, nonempty subset $V' \subseteq V$ and positive integer k, let $BestSubpart(V')$ denote an optimum solution of our unconstrained Problem 4 (that is, a maximizer of $\max\{\sum_{X\in\mathcal{X}}(k - \varrho_w(X)) : \mathcal{X}$ is a subpartition of $V'\}$). Note that $BestSubpart(V)$ always has at least one member (since the subpartition $\{V\}$ is better than the empty subpartition).

Fortunately, the family of insolid sets has a nice structure: as observed in [1], the family of insolid sets forms a subtree hypergraph, defined as follows. (Actually, they proved that the family of solid sets - the family of all insolid or outsolid sets - forms a subtree hypergraph, and the subtree representation can be found in polynomial time. Here we only need the property for the family of insolid sets.)

Definition 2. A hypergraph $H = (V, \mathcal{E})$ is called a **subtree hypergraph** if there exists a tree T spanning the node set V such that every hyperedge in $\mathcal{E}$ induces a subtree of T. The tree T is called a **basic tree** (or representative tree) for the hypergraph H.

Bárász, Becker and Frank proved that a representative tree for insolid sets exists, and can be found in polynomial time. The subtree hypergraph property and its polynomial time construction is quite surprising, given the fact that the number of insolid sets might be exponential.

Theorem 4.5 (Bárász, Becker, Frank [1])**.** *The family $\mathcal{F}_{in} = \mathcal{F}_{in}(D_w)$ of in-solid sets of an arc-weighted digraph $D_w = (V, A, w)$ is a subtree hypergraph. The representative tree can be found in polynomial time in the size of the digraph.*

For an arc-weighted digraph D_w, let $T_{in} = T_{in}(D_w)$ denote a representative tree for the family $\mathcal{F}_{in}$ of insolid sets. In fact we will only need to solve Problem 4 for a set V' that induces a subtree of T_{in}. Observe however that this is not a restriction: if $T_{in}[V']$ is not connected then solving Problem 4 for the components of $T_{in}[V']$ and taking the union of the obtained subpartitions gives a solution for V'. Therefore we may assume in Problem 4 that V' induces a subtree of T_{in}.

Unfortunately, enumerating all the insolid sets is not possible, because there can be exponentially many of them (an example can be found in [6]). However, in the case

of insolid sets, the digraph itself provides a succint representation, thus we can query $\mathcal{F}_{in}$ through certain oracles using the likes of minimum cut algorithms.

We cite the following theorem that characterizes the optimum in Problem 4 for an arbitrary subtree hypergraph. (Note that the notion of subtree hypergraphs is very general, and thus this min-max is outsde of the realm of efficient algorithms, because a subtree hypergraph may not be given as part of the input. The theorem holds anyway, and we will apply in a way that it also applies a polynomial running time.) For sake of completeness, we also provide a proof and algorithm here.

Theorem 4.6 (Frank [3]). *If $\mathcal{E} \subseteq 2^{V'}$ is a subtree hypergraph and $val : \mathcal{E} \to \mathbb{R}_+$ is a weight function then*

$$\max\{\sum_{e\in\mathcal{E}'} val_e : \mathcal{E}' \text{ is a collection of disjoint members of } \mathcal{E}\} = \tag{1}$$

$$\min\{\sum_{v\in V} y_v : y : V' \to \mathbb{R}, y \geq 0, \sum_{v\in e} y_v \geq val_e \text{ for every } e \in \mathcal{E}\}. \tag{2}$$

Proof. We give an algorithmic proof of this theorem. Consider the following LP problem.

$$\max \sum_{e\in\mathcal{E}} val_e x_e \tag{3}$$

$$x : \mathcal{E} \to \mathbb{R}, x \geq 0, \tag{4}$$

$$\sum_{e:v\in e} x_e \leq 1 \text{ for every } v \in V'. \tag{5}$$

This LP is a relaxation of the maximum weight matching problem given in (1), while its LP dual is the minimization problem (2). Weak duality shows that the maximum in the theorem cannot be larger than the minimum. For the proof we need to show that the primal LP has an integer optimum solution for every weight function val. This is shown by the following dynamic programming algorithm. A sketch of the below algorithm goes as follows: Initially, we set node weights y_v to be very large. We specify a node r to be the root of representative tree T, and start scanning the tree taking the leaves first. When a node v is scanned, its weight y_v is lowered as much as possible to maintain feasibility. When all nodes are scanned, $\mathcal{E}'$ is set up by tight sets, starting with one covering the node with positive weight closest to the root node, and recursively, adding tight sets for the remaining components. (A set $e \in \mathcal{E}$ is said to be tight with respect to the given feasible dual solution y if $\sum_{v:v\in e} y_v = val_e$.) The algorithm is detailed below.

Algorithm MaxWeightMatching($V', T, \mathcal{E}, val$)
begin
INPUT: A tree T on node set V', a family $\mathcal{E} \subseteq 2^{V'}$ of subtrees of T, and a weight function $val : \mathcal{E} \to \mathbb{R}$ ($\mathcal{E}$ and val are available via certain oracles: see later).
OUTPUT: A collection $\mathcal{E}'$ of disjoint members of $\mathcal{E}$ and a dual solution y of the LP Problem (3)-(5) so that $\sum_{e\in\mathcal{E}'} val_e = \sum_{v\in V'} y_v$.
1.1. Find some value M with $M > \max_{e\in\mathcal{E}} val_e$.

1.2. Initialize $y_v = M$ for every $v \in V'$.
1.3. Fix an arbitrary node $r \in V'$ and orient T out of r to get $\vec{T}$.
1.4. Let $r = v_1, v_2, \ldots, v_n$ be an order of the nodes of V so that $v_i v_j \in \vec{T}$ implies that $i < j$. Let V_i be the node set of the subtree of $\vec{T}$ rooted at v_i for every i.
1.5. For every $i = n, n-1, \ldots, 1$
1.6. Decrease y_{v_i} as much as possible so that y remains feasible for Problem (2): let e_i be a minimizer of $\min\{y(e) - val_e : v_i \in e \in \mathcal{E}, e \subseteq V_i\}$ and let $y_{v_i} = \max(val_{e_i} - y(e_i - v_i), 0)$ (note that e_i becomes tight, if y_{v_i} stays positive).
1.7. EndFor
1.8. Let $i = 1$, $S = \emptyset$ and $\mathcal{E}' = \emptyset$. //Throughout $S \subseteq V$ and $\mathcal{E}' \subseteq \mathcal{E}$
1.9. While $i \leq n$ do
1.10. If $y_{v_i} > 0$ and $v_i \notin S$ then
1.11. Let $\mathcal{E}' := \mathcal{E}' + \{e_i\}$ and $S := S \cup e_i$
1.12. $i := i + 1$
1.13. EndWhile
1.14. Output $\mathcal{E}'$ and y.
end

The algorithm clearly proves the minmax theorem. In order to implement it in polynomial time in $n = |V'|$ we have to provide the subroutines needed in Steps 1.1 and 1.6. □

Oracles for insolid sets Next, we analyze this algorithm for the solution of Problem 4 to show that it can be implemented in polynomial time when $\mathcal{E}$ is the family of insolid sets and $val_e = k - \varrho_{D_w}(e)$. For that, we need to establish subroutines for steps 1.1 and 1.6 that have running time polynomial in the size of the graph.

It is easy to realize Step 1.1 of Algorithm MaxWeightMatching: $M = k + 1$ will be a good choice.

Step 1.6 can be realized with minimum cut computation as follows. Let $V_i = V(T_{v_i})$ be the node set of the subtree of T rooted at v_i and define a weighted digraph $D'_{w'} = (V + s, A', w')$ as follows: add a new node s to D, introduce an arc sv from s to every $v \in V_i$ of weight $w'(sv) = y_v$, and for an arc $uv \in A(D)$ we define $w'(uv) := w(uv)$. Then by a minimum cut algorithm we find an inclusionwise minimal minimizer of $\min\{\varrho_{D_{w'}}(Z) : v_i \in Z \subseteq V_i\}$. We can solve this problem based on a minimum cut computation, that is, finding an inclusionwise minimum cut between v_i and $\{s\} \cup (V - V_i)$ having minimum weight. We claim that either $\min\{\varrho_{D_{w'}}(Z) : v_i \in Z \subseteq V_i\} \geq y_{v_i}$, or this minimum $\min\{\varrho_{D_{w'}}(Z) : v_i \in Z \subseteq V_i\}$ attains with some set Z that is insolid in D_w. To prove this, consider a set Z attaining the minimum. Then, by Claim 4.2, there is an insolid set $Z' \subseteq Z$ such that $\varrho_{D_w}(Z) \leq \varrho_{D_w}(Z')$. If $v_i \notin Z'$, then we set $y_{v_i} = 0$ in step 1.6. Otherwise Z' is an insolid minimizer, and in 1.6 we set $e_i = z'_i$ and $y_e = val_{Z'_i} - y(Z'_i - v_i)$. This decreases the value of y_{v_i} by $\varrho_{D_w}(Z')$, and thus Z'_i becomes tight (when y_{v_i} stays positive).

This finally proves that Problem 4 can be solved in polynomial time.

4.2 Enforcing the size requirement

For an arbitrary arc-weighted digraph $D_w = (V, A, w)$, nonempty subset $V' \subseteq V$ and positive integer k, recall that $BestSubpart(V')$ denotes an optimum solution of our unconstrained Problem 4 (that is, a maximizer of $\max\{\sum_{X\in\mathcal{X}}(k - \varrho_w(X)) : \mathcal{X}$ is a subpartition of $V'\}$). This can be found in polynomial time by the previous section: for a set V' that induces a subtree of T_{in} we have $BestSubpart(V')$ directly in the output of $MaxWeightMatching(V', T_{in}[V'], \mathcal{F}_{in}[V'], k - \varrho_{D_w})$.

Algorithm $BestConstrSubpart(D_w, k)$
begin
INPUT: An arc-weighted digraph $D_w = (V, A, w)$ and a positive integer k.
OUTPUT: A maximizer of $\max\{\sum_{X\in\mathcal{X}}(k - \varrho_w(X)) : |\mathcal{X}| \geq 2, \mathcal{X}$ is a subpartition of $V\}$
1.1. If the subpartition $BestSubpart(V)$ has at least 2 members then output this and STOP.
1.2. Let T_{in} be a basic tree of D_w. Let $\mathcal{E}' = \emptyset$.
1.3. For every edge e of T consider at most 3 candidates defined as follows.
1.4. Let V_1 and V_2 be the node sets of the 2 components of $T - e$.
1.5. For $i = 1, 2$, let X_i be an inclusionwise minimal minimizer of $\min\{\varrho_{D_w}(X) : \emptyset \neq X \subseteq V_i\}$ (which can be found with minimum cut computation).
1.6. For both $i = 1, 2$, let $\mathcal{P}_i = BestSubpart(V_i)$.
1.7. Candidate 0 is $\{X_1, X_2\}$.
1.8. If $|\mathcal{P}_1| \geq 1$ then Candidate 1 is $\mathcal{P}_1 \cup \{X_2\}$.
1.9. If $|\mathcal{P}_2| \geq 1$ then Candidate 2 is $\mathcal{P}_2 \cup \{X_1\}$.
1.10. Output the best of the candidates above.
end

Theorem 4.7. *Algorithm BestConstrSubpart is correct.*

Proof. Clearly, the Algorithm outputs a feasible solution (that is, a subpartition with at least 2 members), since all its candidates are feasible. Let $\mathcal{P}$ be an optimal insolid subpartition. The proof is complete if we show that one of the candidates is at least as good as $\mathcal{P}$. Choose an edge e of T_{in} so that one of the components (call it V_1) of $T - e$ contains only one member of $\mathcal{P}$. If $\mathcal{P}$ has only 2 members then clearly Candidate 0 for the edge e is not worse than $\mathcal{P}$, so we can assume that $|\mathcal{P}| \geq 3$. Let V_2 be the other component of $T - e$. If $BestSubpart(V_2)$ has at least 1 member then Candidate 2 is not worse than $\mathcal{P}$. But if $BestSubpart(V_2)$ has no members then Claim 4.4 gives that $\varrho_{D_w}(X) \geq k$ for every nonempty $X \subseteq V_2$, therefore Candidate 0 should not be worse than $\mathcal{P}$, finishing the proof. □

The following example shows that even if $BestSubpart(V)$ is not feasible (that is, it has only 1 member), $BestConstrSubpart(V)$ might have more than 2 members: let $k = 4$ and the digraph be directed circuit of size 3, where every arc has weight 3.

5 Analysis

In the algorithm designed above to solve Problem 2 and 3, we apply two steps. In the first step we determine a basic tree representation of the in-solid sets, which according to [1] can be done in $n^3S(n,m)$ time, where n is the number of nodes and m is the number of edges in D, and $S(n,m)$ denotes the time complexity of finding a minimum cut. In the second step, we apply our algorithm for Problem 4 for $O(n^2)$ different subsets U of V, which are determined by removing two edges from the basic tree. (There are $O(n^2)$ different pairs of edges of the basic tree.) For any given subset U, Problem 4 is solved in time $nS(n+1,3m)$, since we apply a minimum cut algorithm to determine the value of $y(v)$ for all nodes v in U by a minimum cut computation for graphs of at most $n+1$ nodes and at most $m+kn$ arcs. (To see this, note that only one node s is added to the graph, and the number of arcs added is equal to the sum of $y(v)$'s, which is equal to the sum $\sum_{X\in\pi}(k-\varrho(X))$ over a partition π of $U+s$.) Note that $kn \leq 2m$, since otherwise there may be no k disjoint arborescences. The total running time for the algorithm amounts to $n^3S(n+1,3m)$. Thus, by using Orlin's algorithm [7] for minimum cut computations, the running time is bounded by $O(n^4m)$.

References

[1] Mihály Bárász, Johanna Becker, and András Frank, *An algorithm for source location in directed graphs*, Oper. Res. Lett. **33** (2005), no. 3, 221–230.

[2] Attila Bernáth and Gyula Pap, *Blocking optimal arborescences*, Integer Programming and Combinatorial Optimization, Springer, 2013, pp. 74–85.

[3] András Frank, *Some polynomial algorithms for certain graphs and hypergraphs*, Proceedings of the Fifth British Combinatorial Conference, 1975, pp. 211–226.

[4] András Frank, *On disjoint trees and arborescences*, Algebraic Methods in Graph Theory, Colloquia Mathematica Soc. J. Bolyai, vol. 25, 1978, pp. 159–169.

[5] András Frank, *Covering branchings*, Acta Scientiarum Mathematicarum (Szeged) **41** (1979), no. 77-81, 3.

[6] Hiro Ito, Kazuhisa Makino, Kouji Arata, Shoji Honami, Yuichiro Itatsu, and Satoru Fujishige, *Source location problem with flow requirements in directed networks*, Optimization Methods and Software **18** (2003), no. 4, 427–435.

[7] James B. Orlin, *Max flows in o(nm) time, or better*, STOC, 2013, pp. 765–774.

[8] Alexander Schrijver, *Combinatorial optimization: polyhedra and efficiency*, vol. 24, Springer Verlag, 2003.