Algorithmica (2011) 61:898–922
DOI 10.1007/s00453-009-9377-1

An Efficient Scaling Algorithm for the Minimum Weight Bibranching Problem

Maxim A. Babenko

Received: 23 December 2008 / Accepted: 16 November 2009 / Published online: 16 December 2009

Abstract Let $G = (VG, AG)$ be a digraph and let $S \sqcup T$ be a bipartition of VG. A *bibranching* is a subset $B \subseteq AG$ such that for each node $s \in S$ there exists a directed s–T path in B and, vice versa, for each node $t \in T$ there exists a directed S–t path in B.

Bibranchings generalize both branchings and bipartite edge covers. Keijsper and Pendavingh proposed a strongly polynomial primal-dual algorithm that finds a minimum weight bibranching in $O(n'(m + n \log n))$ time (where $n := |VG|$, $m := |AG|$, $n' := \min(|S|, |T|)$).

Assuming that arc weights are integers we develop a weight-scaling algorithm of time complexity $O(m\sqrt{n}\, \log n \log(nW))$ for the minimum weight bibranching problem (where W denotes the maximum magnitude of arc weights).

Keywords Branching · Bipartite edge cover · Weight scaling · Primal-dual algorithm · Blocking augmentation

1 Introduction

In a digraph G, the sets of nodes and arcs are denoted by VG and AG, respectively. A similar notation is used for paths, cycles, and etc.

Consider a digraph G and a fixed bipartition $S \sqcup T$ of VG. A subset $B \subseteq AG$ is called a *bibranching* if the following conditions are met:

– for each $s \in S$, set B contains a directed path from s to some node in T;
– for each $t \in T$, set B contains a directed path from some node in S to t.

Supported by RFBR grant 06-01-00122.

M.A. Babenko (✉)
Dept. of Mechanics and Mathematics, Moscow State University, Yandex LLC, Russia
e-mail: max@adde.math.msu.su

Bibranchings were introduced in [14]. In the present paper we study the *minimum weight bibranching problem*, which is as follows:

(BB) Given G, S, T, and *arc weights* $w\colon AG \to \mathbb{R}_+$, find a bibranching B whose weight $w(B)$ is minimum.

Hereinafter we assume that every real-valued function $f\colon U \to \mathbb{R}$ is additively extended to the family of all subsets of U (denoted by 2^U) by $f(A) := \sum_{a\in A} f(a)$. In particular, $w(B)$ denotes the sum of weights of arcs in B.

Minimum weight bibranchings provide a common generalization to a pair of well-known combinatorial problems. Firstly, if $S = \{s\}$ then (BB) asks for a minimum weight *s-branching* (a directed tree rooted at s that covers all nodes of G). An $O(m + n\log n)$ algorithm for the latter task is known [9] (here $n := |VG|$ and $m := |AG|$; we are assuming throughout the paper that $n \le m \le n^2$). Another special case arises when digraph G only contains S–T arcs (but no T–S, S–S, or T–T arcs). Then, the definition of a bibranching reduces to that of a *bipartite edge cover*. The minimum weight bipartite edge cover problem (call it (EC) for brevity) looks harder: no strongly polynomial $O(mn)$-algorithm is known so far. Problem (EC) can be solved by finding a maximum weight bipartite matching in $O(n'(m + n\log n))$ time [6] (where $n' := \min(|S|, |T|)$). Keijsper and Pendavingh [13] generalized the latter shortest-path augmentation method to solve (BB) in the same time bound.

On the other hand, the notion of a bibranching is special case of a *directed cut cover* (also known as a *dijoin*). The latter is a subset of arcs whose contraction makes a digraph strongly connected, see [15, Sect. 55.1]. Frank studied the minimum weight dijoin problem and gave an $O(n^5)$-time primal-dual algorithm [5]. Later, this bound was improved to $O(mn^2)$ by Gabow [7]. For the case of non-negative integer weights Gabow also devised an $O(\min(m^{1/2}, n^{2/3})\, nm\log(nW))$-time algorithm [8] (hereinafter W denotes the maximum magnitude of arc weights).

In general, many optimization problems can be solved faster if weights are known to be integral. The corresponding algorithms (e.g. the above mentioned Gabow's minimum weight dijoin algorithm) are based on scaling techniques and achieve time bounds that are, in a sense, better than their strongly-polynomial counterparts. Another example is a scaling algorithm for bipartite matching problems [11], which runs in $O(m\sqrt{n}\,\log(nW))$ time. The latter approach can also be adopted to solve (EC) and leads to the algorithm with the same time bound.

Similar ideas are also applicable to general min-cost flow problems [10]. However, when the structure of dual solutions becomes non-trivial (i.e. when one needs exponentially many dual variables) the algorithm and its complexity analysis may become much more involved. A good example is the minimum weight perfect matching problem in general (non necessarily bipartite) graphs, which is solved by Gabow and Tarjan in $O(m\sqrt{n\alpha(m,n)\log n}\,\log(nW))$ time [12] (where α stands for the inverse Ackermann function).

Since (BB) involves solving a linear program with inequalities corresponding to all possible subsets of S and T, our approach is similarly involved. Assuming that arc weights w are non-negative integers not exceeding W we present a weight scaling algorithm for (BB) that runs in $O(m\sqrt{n}\,\log n\log(nW))$ time. It is based on the general notion of the approximate optimality (see, e.g., [10]), an augmentation procedure similar to [13], and attracts some additional combinatorial ideas to deal with

dual solutions during scaling steps. Also, a variant of the blocking augmentation technique [3] is employed.

Note that the complexity of our algorithm coincides (up to a logarithmic factor) with that of the best known scaling algorithm [11] for solving a special case of (BB), namely (EC). Also, our algorithm is faster than Gabow's one [8], which is not surprising since the latter deals with a more general problem.

The rest of the paper is organized as follows. Section 2 gives the needed formal background and introduces the linear programming formulation of (BB). Sections 3 and 4 outline a high-level description of the algorithm. Section 5 bounds the number of primal and dual steps performed by the algorithm. Section 6 discusses the details of an efficient implementation of the algorithm. Section 7 summarizes the results and describes a number of open questions.

An extended abstract of this paper [1] earlier appeared at ISAAC 2008.

2 Preliminaries

First, we need some additional definitions and notation. Let G be a digraph and X be a subset of nodes. Then $\delta^{\text{in}}_G(X)$ (resp. $\delta^{\text{out}}_G(X)$ and $\gamma_G(X)$) denotes the set of arcs entering X (resp. leaving X and having both endpoints in X). When it is clear from the context which digraph G is meant, it is omitted from the notation. Also, when $X = \{v\}$ write just $\delta^{\text{in}}(v)$ (resp. $\delta^{\text{out}}(v)$) instead of $\delta^{\text{in}}(\{v\})$ (resp. $\delta^{\text{out}}(\{v\})$).

For a digraph G and a subset $X \subseteq VG$ let $G[X]$ denote the subgraph of G *induced by* X (i.e. the digraph obtained from G by removing all nodes in $VG - X$).

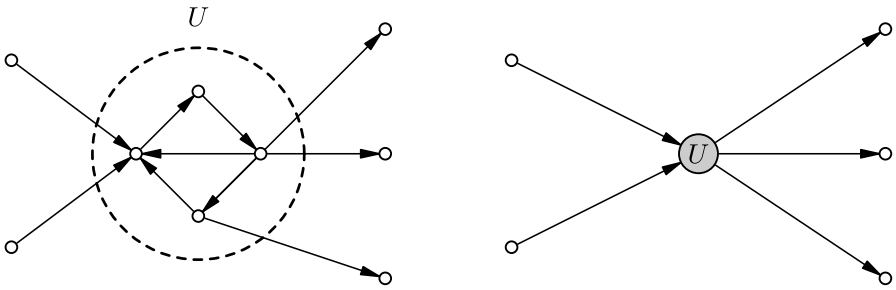
By an *in-* (resp. *out-*) *forest* we mean an acyclic set of arcs F such that for each node v at most one arc in F leaves (resp. enters) v. If no arc in F leaves (resp. enters) node v then v is said to be a *root* of F. An in- (resp. out-) forest with a single root node is called an *in-* (resp. *out-*) *tree*.

Consider a bipartition $S \sqcup T$ of VG. For a subset $X \subseteq S$ we put $\delta(X) := \delta^{\text{out}}(X)$, similarly for $X \subseteq T$ we put $\delta(X) := \delta^{\text{in}}(X)$. If $a \in \delta(X)$ then arc a is said to *cover* X. For a set $A \subseteq AG$, the *set of nodes covered by* A is defined as the union of the sets of nodes covered by the individual elements of A. Clearly, (BB) prompts for a minimum weight subset $B \subseteq AG$ that covers every subset in $2^S \cup 2^T$.

Let us introduce an important notion of *contraction*. To *contract* a set $X \subseteq VH$ in a digraph H one replaces nodes in X by a single *complex node* (also denoted by X). Arcs in $\gamma(VH - X)$ are not affected, arcs in $\gamma(X)$ are dropped, and arcs in $\delta^{\text{in}}(X)$ (resp. $\delta^{\text{out}}(X)$) are redirected so as to enter (resp. leave) the complex node X. The resulting digraph is denoted by H/X. See Fig. 1 for an example.

Note that H/X may contain multiple parallel arcs. We identify arcs in H/X with their pre-images in H. If H' is obtained from H by an arbitrary sequence of contractions, then $H' = H/X_1/\ldots/X_k$ for a certain family of disjoint subsets $X_1, \ldots, X_k \subseteq VH$ (called the *maximal contracted sets*). Each node in H' corresponds to a subset of nodes in H: nodes $v \in VH - (X_1 \cup \ldots \cup X_k)$ are called *simple* and correspond to singletons $\{v\}$. Other nodes are complex and correspond to the sets X_i.

For a set $A \subseteq AG$, we write A_{SS}, A_{ST}, and A_{TT} to denote the sets of S–S, S–T, and T–T arcs in A, respectively. Note that any minimum weight bibranching need not



(a) Digraph H. (b) Digraph H/X.

Fig. 1 Contraction

contain T–S arcs (as their removal preserves the required connectivity and may only decrease the total weight). Hence, we assume throughout the paper that G contains no T–S arcs.

We call a bibranching B *basic* if B is inclusion-wise minimal, i.e. no proper subset $B' \subset B$ is a bibranching. The following observations are easy:

Lemma 1 *Any basic bibranching contains at most n arcs.*

Proof Let B be a basic bibranching. Suppose that B_{SS} contains a directed cycle C. There must be an arc $a_0 \in B$ that leaves VC, let v_0 be its tail node. Next, let a_1 be the arc in C that leaves v_0. Now after resetting $B := B - \{a_1\}$ set B remains a bibranching, which is a contradiction. Similar reasoning also applies to B_{TT}.

Hence, B_{SS} and B_{TT} are directed forests. Let R_S and R_T denote the sets of their roots, respectively. Then, $|B_{SS}| = |S| - |R_S|$ and $|B_{TT}| = |T| - |R_T|$.

Set B_{ST} covers all nodes in $R_S \cup R_T$. Moreover, B_{ST} is an inclusion-wise minimal arc set with the above property. For each node $v \in R_S \cup R_T$ one can fetch an arbitrary arc a_v from B_{ST} that covers v. The constructed arc set $\{a_v \mid v \in R_S \cup R_T\}$ covers all nodes in $R_S \cup R_T$ and, hence, coincides with B_{ST}. Therefore, $|B_{ST}| \leq |R_S| + |R_T|$.

Summing the above, one gets $|B| \leq (|S| - |R_S|) + (|T| - |R_T|) + |R_S| + |R_T| = |S| + |T| = n$, as required. □

Lemma 2 *For each bibranching B a basic bibranching $B' \subseteq B$ can be found in $O(m)$ time.*

Proof Firstly, we compute the strongly-connected components of B_{SS} in $O(m)$ time, see [2]. For each component C in B_{SS} there must be at least one arc a_0 that leaves VC. Let v_0 be the tail node of a_0. Applying the depth-first search we construct a spanning in-tree T in C (i.e. $VT = VC$, $AT \subseteq AC$) rooted at v_0 and reset $B := (B - AC) \cup AT$.

Once this is done for each component C, set B_{SS} is decycled and B remains a bibranching. Similar procedure is applied to B_{TT}. Totally these manipulations take $O(m)$ time. Now B_{SS} and B_{TT} are directed forests.

Next, if a node $v \in S$ is covered by arcs from both B_{SS} and B_{ST}, then the former arc can be removed from B_{SS}. Similarly, we process the set B_{TT}. Now B_{SS} and B_{TT}

are inclusion-wise minimal (that is, no arc can be removed from these sets without breaking the required connectivity of B).

It remains to deal with B_{ST}. To this aim, we scan its elements one after another. Let (u, v) be the current arc to be examined ($u \in S$, $v \in T$). Node u (resp. v) is a root in B_{SS} (resp. in B_{TT}). If both u and v are covered by at least two arcs in B_{ST} then we remove (u, v) from B.

To carry out this cleanup efficiently, for each root node v we keep a counter that is equal to the number of arcs in B_{ST} covering v. Using these counters, processing each arc in B_{ST} takes $O(1)$ time.

Clearly, the resulting set B is inclusion-wise minimal. □

Now let us deal with the linear programming formulation of (BB). Consider the following program:

$$\begin{aligned} \text{minimize} \quad & \sum (w(a)x(a) : a \in AG) \\ \text{subject to} \quad & x : AG \to \mathbb{R}_+, \\ & x(\delta(X)) \geq 1 \quad \text{for each } X \in 2^S \cup 2^T. \end{aligned} \tag{2.1}$$

The program dual of (2.1) is:

$$\begin{aligned} \text{maximize} \quad & \sum \left(\pi(X) : X \in 2^S \cup 2^T\right) \\ \text{subject to} \quad & \pi : 2^S \cup 2^T \to \mathbb{R}_+, \\ & w_\pi(a) \geq 0 \quad \text{for each } a \in AG. \end{aligned} \tag{2.2}$$

Here $w_\pi := w - \vartheta\pi$ are the *reduced weights* of arcs w.r.t. π, and the function $\vartheta\pi : AG \to \mathbb{R}_+$ is defined by

$$\vartheta\pi(a) := \sum (\pi(X) : a \text{ covers } X). \tag{2.3}$$

It is known [14] (see also [15, Sect. 54.6]) that (2.1) describes the upper convex hull of the incidence vectors of all bibranchings in G. Hence, finding a bibranching of the minimum weight (under the assumption $w \geq 0$) amounts to finding a 0,1-solution to (2.1).

Weak duality for (2.1) and (2.2) implies that $\sum_a w(a)x(a) \geq \sum_X \pi(X)$ holds for every pair of admissible solutions x and π. By strong duality, the latter turns into equality if x and π are optimal. Moreover, (2.1) is known to be *totally dual integral* (see [15, Sect. 54.6]), that is, if all weights w are integers then there exists an optimal integer solution to (2.2). Hence, the polyhedron determined by (2.1) is integral.

The complementary slackness conditions for (2.1) and (2.2) (giving rise to an optimality criterion for solutions x and π) are viewed as:

(CS1) $x(a) > 0$ for $a \in AG$ implies $w_\pi(a) = 0$.
(CS2) $\pi(X) > 0$ for $X \in 2^S \cup 2^T$ implies $x(\delta(X)) = 1$.

For a set $B \subseteq AG$ and a function $\pi : 2^S \cup 2^T \to \mathbb{R}_+$ we say that B is π*-consistent* if $\pi(X) > 0$ implies $|B \cap \delta(X)| \leq 1$ for each $X \in 2^S \cup 2^T$. Consistency is closely

related to the complementary slackness conditions, in particular, if B is a bibranching then its π-consistency is just (CS2).

Lemma 3 *For an arbitrary function $\pi : 2^S \cup 2^T \to \mathbb{R}_+$ and a set $B \subseteq AG$ one has*

$$\vartheta\pi(B) \geq \sum (\pi(X) : B \text{ covers } X) .$$

Additionally, if B is π-consistent then the above inequality turns into equality.

Proof Taking (2.3) into account, one needs to prove the following:

$$\sum_{a \in B} \sum (\pi(X) : a \text{ covers } X) \geq \sum (\pi(X) : B \text{ covers } X) .$$

The latter is obvious since for each term $\pi(X)$ occurring on the right there must be at least one such term on the left. Moreover, equality is only possible if for each subset $X \in \text{supp}(\pi)$ that is covered by B, X is actually covered by a unique arc in B. This is exactly π-consistency of B. □

3 Algorithm

Recall that a family $\mathcal{F}$ of subsets is called *laminar* if for all $X, Y \in \mathcal{F}$ either $X \subseteq Y$, or $Y \subseteq X$, or $X \cap Y = \emptyset$.

The algorithm maintains a laminar family $\mathcal{F} \subseteq 2^S \cup 2^T$ and a function $\pi : 2^S \cup 2^T \to \mathbb{Z}_+$. For $X \in \mathcal{F}$, the *shell* $S(X)$ of X is the graph obtained from $G[X]$ by contracting all proper maximal subsets $Y \subset X$, $Y \in \mathcal{F}$. Let $\bar{G}$ denote the digraph obtained by contracting all maximal sets of $\mathcal{F}$ in G. Put $\bar{S}$ (resp. $\bar{T}$) to be the image of S (resp. T) in $\bar{G}$ under these contractions. Let $S^0_\pi(X)$ denote the subgraph of $S(X)$ consisting of arcs a with $w_\pi(a) = 0$.

For a set of nodes Y in $\bar{G}$ (or in $S(X)$ for $X \in \mathcal{F}$) we write $\tilde{Y}$ to denote the corresponding pre-image subset in VG. When $Y = \{y\}$ we write just $\tilde{y}$ instead of $\widetilde{\{y\}}$.

We introduce the following set of properties:

(D1) $\pi(X) > 0$ and $|X| > 1$ imply $X \in \mathcal{F}$.
(D2) $S^0_\pi(X)$ has a directed Hamiltonian cycle for each $X \in \mathcal{F}$.
(D3) $w_\pi(a) \geq 0$ for each $a \in AG$.

Property (D1) claims that every non-singleton subset that is assigned a positive dual is contracted. Property (D2) introduces an additional structural property of π that is necessary to uncontract $\bar{G}$ back into G and restore the bibranching. Finally, property (D3) states that π is a feasible solution to (2.2).

Next, the algorithm maintains a subset $\bar{B} \subseteq A\bar{G}$, which will be referred to as a *partial bibranching*. We associate the following properties with $\bar{B}$:

(P1) Set $\bar{B}_{SS}$ (resp. $\bar{B}_{TT}$) forms a directed in- (resp. out-) forest in digraph $G[\bar{S}]$ (resp. in digraph $G[\bar{T}]$).
(P2) If $a \in \bar{B}_{SS} \cup \bar{B}_{TT}$ then $w_\pi(a) = 0$; if $a \in \bar{B}_{ST}$ then $w_\pi(a) \leq 1$.

(P3) If $v \in V\bar{G}$ is covered by more than one arc in $\bar{B}$ then v is a simple node and $\pi(\tilde{v}) = 0$ (cf. (CS2)).

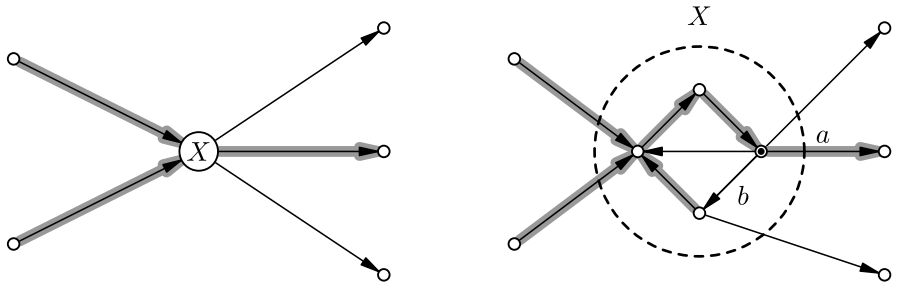
Property (P1) is required mostly by technical reasons that will become evident later. Property (P2) may be regarded as a relaxation of (CS1). Property (P3) corresponds to (CS2).

In what follows, we shall use the procedure called EXPAND. It relies on property (D2), expands a maximal contracted subset $X \in \mathcal{F}$, and simultaneously extends the current partial bibranching $\bar{B}$.

For simplicity's sake suppose $X \subseteq S$, the other case is symmetric. Firstly, digraph $\bar{G}$ is partly uncontracted by undoing the contraction of X. Let X' denote the set of nodes arising from X during this uncontraction. Note that if there is an arc $a \in \bar{B}$ that covers v before the uncontraction then the latter arc is unique by (P3). If a does not exist, we extend $\bar{B}$ by adding all arcs of the Hamiltonian cycle C mentioned in (D2) except one, which is chosen arbitrarily. This way, the resulting $\bar{B}$ covers all nodes in X' except one. If a exists then, again, all arcs of C except one, call it b, are added to $\bar{B}$. The latter arc b shares its tail node with a. The resulting $\bar{B}$ covers all nodes in X'. Clearly, in both cases the number of uncovered nodes does not change during EXPAND. Also, properties (P1)–(P3) are maintained. Figure 2 shows an example.

The algorithm for solving (BB) employs *bit scaling* and works as follows. Let $w_0 \colon AG \to \mathbb{Z}_+$ denote the input weight function. A certain current weight function $w \colon AG \to \mathbb{Z}_+$ is maintained. Initially $w := 0, \pi := 0, \mathcal{F} := \emptyset$. In particular, no subset is contracted in G, so $\bar{G} = G$. Also, $\bar{B}$ is initialized with an arbitrary bibranching in $\bar{G}$ $(= G)$ obeying property (P1) (the one obtained from AG via Lemma 2 will do). In case $\bar{B}$ does not exist, the algorithm halts.

Each scaling step takes a weight function w from the previous iteration, a bibranching $\bar{B} \subseteq A\bar{G}$ in $\bar{G}$, a function π, and a collection $\mathcal{F}$ altogether obeying properties (P1)–(P3), (D1)–(D3). The i-th $(1 \le i \le l)$ scaling step deals with the weights $w(a) := \lfloor w_0(a)/2^{l-i} \rfloor$, $a \in AG$. Here $l := \lfloor \log W \rfloor + 1$ denotes the length of the binary representation of the largest arc weight W. Less formally, this means that at every next scaling step one more bit of the binary representation of the original weight function w_0 is taken in account.



(a) Before uncontraction. (b) After uncontraction.

Fig. 2 Application of EXPAND. Arcs in $\bar{B}$ are grayed; the common tail of arcs a and b is marked

At the beginning of each scaling step the previous weights $w(a)$, $a \in AG$, are doubled and some of them are increased by 1 (namely, those having 1 at the corresponding position of the binary representation of $w_0(a)$). Changing weights w may lead to violation of the above properties (P1)–(P3), (D1)–(D3) so the goal of the scaling step is to restore them. The necessary details will be given in Sect. 4.

Weights w are iteratively scaled, as explained above, until achieving the equality $w = w_0$. Totally it takes l scaling steps. Next, we put $t := \lfloor \log n \rfloor + 1$ and perform t additional scaling steps, doubling w each time. Finally, the algorithm applies Lemma 4 to construct the final bibranching in G.

Lemma 4 *Suppose properties* (D1)–(D3), (P1)–(P3) *hold for* $\mathcal{F}$, π, *and* $\bar{B}$. *Also, suppose each node in* $\bar{G}$ *is covered by some arc in* $\bar{B}$. *Then, there exists a* π*-consistent basic bibranching* B *in* G *such that*: $w_\pi(a) = 0$ *for each* $a \in B_{SS} \cup B_{TT}$ *and* $w_\pi(a) \leq 1$ *for each* $a \in B_{ST}$.

Proof Property (P1) implies that $\bar{B}$ is acyclic. Also, $\bar{B}$ covers every node in $\bar{G}$, therefore $\bar{B}$ is a bibranching in $\bar{G}$. Applying EXPAND to the elements of $\mathcal{F}$ (in an appropriate order) one gets a π-consistent bibranching B in G. Removing an appropriate subset from it, we get the desired basic bibranching. □

Let us prove that this general scheme is correct. Put $\Pi := \sum_X \pi(X)$ and estimate the weight of an optimal bibranching as follows.

Lemma 5 $w(B) \geq \Pi$ *holds for any bibranching* B *in* G.

Proof By definition, B covers each subset in $2^S \cup 2^T$. Hence, by Lemma 3 and (D3) one has $w(B) = w_\pi(B) + \vartheta\pi(B) \geq \sum_X \pi(X) = \Pi$, as required. □

Lemma 6 *If* B *is a basic* π*-consistent bibranching in* G *such that* $w_\pi(a) \leq 1$ *for all* $a \in B$ *then* $w(B) \leq \Pi + n$.

Proof By Lemmas 1 and 3 one has $w(B) = w_\pi(B) + \vartheta\pi(B) \leq n + \sum_X \pi(X) = n + \Pi$. □

Theorem 1 *The algorithm constructs a minimum weight bibranching.*

Proof Let $\bar{B}$, w, and π denote the corresponding objects after the last scaling step. Put B to be a basic bibranching obtained from $\bar{B}$ by Lemma 4. Let $B_{\min}$ be a minimum weight bibranching (w.r.t. w or, equivalently, w_0). One may assume by Lemma 2 that $B_{\min}$ is basic. Lemmas 5 and 6 imply that $w(B_{\min}) \geq \Pi$ and $w(B) \leq \Pi + n$ respectively, so $w(B_{\min}) \leq w(B) \leq w(B_{\min}) + n$. Recall that each of the last t scaling steps doubles arc weights. Since all initial weights w_0 are integers, $w(a)$ is divisible by 2^t for each $a \in AG$. Hence, so are $w(B)$ and $w(B_{\min})$. The choice of t implies $n < 2^t$, therefore $w(B) = w(B_{\min})$, so B is optimal. □

4 Scaling Step

Each scaling step consists of the following four stages: doubling stage, shell stage, ST-stage, and TS-stage.

First, the *doubling stage* is executed: arc weights w are multiplied by 2 and some of them are increased by 1, as described in Sect. 3. Also, duals π are doubled. Put $\mathcal{F} := \operatorname{supp}(\pi)$ and $\bar{B} := \emptyset$ (hence, any previous bibranching is discarded). As earlier, let $\bar{G}$ denote the digraph obtained from G by contracting all maximal sets in $\mathcal{F}$. Obviously, properties (D1), (D3), (P1)–(P3) now hold. One needs to solve the following two tasks:

- restore property (D2) by ensuring that each graph $S^0_\pi(X)$, $X \in \mathcal{F}$, contains a directed Hamiltonian cycle;
- construct a bibranching $\bar{B}$ in $\bar{G}$ obeying properties (P1)–(P3).

The *shell stage* is executed to deal with (D2). The algorithm scans the sets in $\mathcal{F}$ choosing an inclusion-wise minimal unscanned set at each iteration. Let $X \in \mathcal{F}$ be the current set to be scanned. Procedure NORMALIZE-SHELL(X) is called to adjust duals π and ensure (D2) for X or remove X from $\operatorname{supp}(\pi)$ and also from $\mathcal{F}$.

Suppose $X \subseteq S$, the case $X \subseteq T$ is analogous. NORMALIZE-SHELL performs a series of iterations similarly to the minimum weight branching algorithm [4, 9].

More precisely, it maintains a directed in-forest F_S containing all nodes of X and consisting of some arcs a with $w_\pi(a) = 0$. Initially the forest is trivial: $VF_S := VS(X)$ and $AF_S := \emptyset$. If $S(X)$ consists of a single node then property (D2) is restored for X, NORMALIZE-SHELL(X) terminates.

Otherwise, an arbitrary tree W in F_S is picked. Let r be the root of W. Suppose that all arcs leaving r (in $S(X)$) have positive reduced weights. Put

$$\mu_1 := \min\left(w_\pi(a) : a \in \delta^{\text{out}}_{S(X)}(r)\right), \qquad \mu_2 := \pi(X), \qquad \mu := \min(\mu_1, \mu_2).$$

Adjust the duals as follows:

$$\pi(\tilde{r}) := \pi(\tilde{r}) + \mu,$$
$$\pi(X) := \pi(X) - \mu.$$

These adjustments decrease the reduced weights of all arcs in $S(X)$ leaving r by μ (see Fig. 3). If one had $\pi(\tilde{r}) = 0$ prior to the adjustment then $\tilde{r}$ gets added to $\operatorname{supp}(\pi)$, in this case the algorithm also adds $\tilde{r}$ to $\mathcal{F}$. By the choice of unscanned sets for NORMALIZE-SHELL, property (D2) holds for $\tilde{r}$ and all its subsets in $\mathcal{F}$ (if any). Set $\tilde{r}$ is also marked as scanned, so NORMALIZE-SHELL is never called for it.

If $\pi(X) = 0$ holds after the adjustment then set X vanishes from $\operatorname{supp}(\pi)$, we remove X from $\mathcal{F}$ and halt NORMALIZE-SHELL(X). We also say that set X *dissolves* during the execution of the shell stage.

Now suppose that there is an arc $a \in \delta^{\text{out}}_{S(X)}(r)$ such that $w_\pi(a) = 0$. Two cases are possible. Firstly, a may connect W with another tree W' in F_S. Then, a is added to F_S thus linking W and W'. Secondly, a may connect r to a node in the very same tree W. In this case, a cycle of arcs with zero reduced weights is discovered. Let Y be

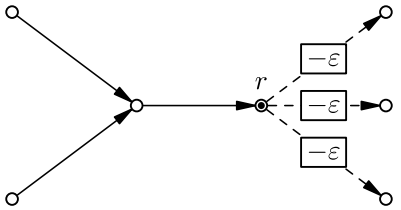


Fig. 3 Dual adjustment during the shell stage. Dashed arcs have positive reduced weights

the set of nodes of this cycle (in $S(X)$). The algorithm contracts Y in $S(X)$, adds $\tilde{Y}$ to $\mathcal{F}$, and marks $\tilde{Y}$ scanned. Note that $\pi(\tilde{Y}) = 0$ holds at this point. Next, the algorithm proceeds by picking a tree W in F_S and repeating the above steps.

Once NORMALIZE-SHELL(X) is called for all sets $X \in \mathcal{F}$ (in an appropriate order), property (D2) gets restored. The normalization procedure for a subset $X \subseteq T$ is the same except for the fact that it considers sets δ^{in} rather than δ^{out}.

The remaining part of the scaling step builds a bibranching $\bar{B}$ in $\bar{G}$ that satisfies properties (P1)–(P3). Firstly, the *ST-stage* is executed. It starts with $\bar{B} = \emptyset$ and applies a certain augmenting path approach aiming to update $\bar{B}$ so that it covers all the subsets of $2^{\bar{S}}$. Next, S and T parts are exchanged and a similar *TS-stage* is executed, thus completing the scaling step.

We shall only describe the ST-stage since the TS-stage is essentially symmetric. However, due to the generic nature of the description we cannot make any additional assumptions about the set $\bar{B}$ at the beginning of the stage. In particular, we cannot assume that $\bar{B}$ is empty.

For the duration of the ST-stage, we impose the following additional property:

(D4) Each maximal contracted set $X \subseteq T$, $X \in \mathcal{F}$ obeys $|X| > 1$ and $\pi(X) > 0$.

If (D4) fails at the beginning of the stage then EXPAND routine is iteratively applied to the maximum contracted subsets $X \subseteq T$ with $\pi(X) = 0$.

Similarly to the shell stage, a directed in-forest F_S obeying $VF_S = \bar{S}$ is maintained in digraph $\bar{G}$. The latter forest obeys the following conditions:

(S1) $w_\pi(a) = 0$ holds for each $a \in AF_S$.
(S2) $w_\pi(a) > 0$ holds for each root node $r \in \bar{S}$ and an $\bar{S}$–$\bar{S}$ arc a leaving r.
(S3) $\bar{B}_{SS} \subseteq AF_S$.

Forest F_S is initially constructed by putting $VF_S := \bar{S}$ and $AF_S := \bar{B}_{SS}$ (the latter set forms a directed in-forest according to (P1)). Next, NORMALIZE-FOREST routine is applied to ensure (S2). The latter works as follows. If (S2) fails for a root node r and an $\bar{S}$–$\bar{S}$ arc a leaving r then two cases are possible, similarly to the shell stage. Firstly, a may connect a tree W of F_S rooted at r with another tree W' in F_S. Then, a is added to F_S thus linking W and W'. Secondly, a may connect r to a node in the very same tree W. In this case, a cycle consisting of arcs with zero reduced weights is discovered. Let Y denote the set of nodes of this cycle (in $\bar{G}$). The algorithm puts $\bar{B} := \bar{B} \setminus \gamma(Y)$, $AF_S := AF_S \setminus \gamma(Y)$, contracts Y in $\bar{G}$, and adds $\tilde{Y}$ to $\mathcal{F}$. Note that at this point $\pi(\tilde{Y}) = 0$ holds.

This completes the description of NORMALIZE-FOREST.

Once forest F_S obeying (S1)–(S3) is ready, we construct an auxiliary digraph H, which will be used for blocking augmentation. Put $VH := V\bar{G}$ and proceed as follows:

- if $a \notin \bar{B}$ is an $\bar{S}$–$\bar{T}$ arc with $w_\pi(a) = 0$ then add a to H as a *forward* arc;
- if $a \in \bar{B}$ is an $\bar{S}$–$\bar{T}$ arc with $w_\pi(a) = 1$ then reverse the direction of a and it to H as a *backward* arc.

A node $v \in \bar{S}$ is said to be *initial* if $\bar{B}$ does not cover v. A node v is called *final* if any of the following cases applies:

(F1) $v \in \bar{T}$ is a simple node obeying $\pi(\tilde{v}) = 0$.
(F2) $v \in \bar{T}$ and v is covered by a $\bar{T}$–$\bar{T}$ arc in $\bar{B}$ (the latter is unique by (P1)).
(F3) $v \in \bar{T}$ and v is not covered by $\bar{B}$.
(F4) $v \in \bar{S}$ and v is an inner node in F_S.
(F5) $v \in \bar{S}$ and v is covered by at least two $\bar{S}$–$\bar{T}$ arcs in $\bar{B}$.

A directed node-simple path in H connecting an initial node to a final node (with all intermediate nodes neither initial nor final) is called *augmenting*.

Note that a node may simultaneously be initial and final. This happens when one has an uncovered node $v \in \bar{S}$ that is an inner node in F_S. In this case node v forms a *trivial* augmenting path of zero length.

The ST-stage consists of an alternating sequence of steps of two types. Firstly, suppose that there exists an augmenting path P in H from an initial node s to a final node t. In this case, a *primal step* is possible. Its goal is to update $\bar{B}$ so as to decrease the number of uncovered nodes in $\bar{S}$ while not increasing the number of uncovered nodes in $\bar{T}$. (Note that we have to make this additional remark regarding the $\bar{T}$ part because we are going to carry out the TS-stage similarly to the ST- one. Hence, we do not want any new uncovered nodes to appear.)

Given an augmenting path P, we replace the current primal solution $\bar{B}$ with the symmetric difference $\bar{B} \triangle A(P)$, where $A(P)$ denotes some set of arcs corresponding to P. The latter set $A(P)$ is constructed as follows. First, we add arcs that correspond to arcs of P (both forward and backward). In case of bipartite matchings and edge covers these are enough, however we still have S–S and T–T arcs that need to be taken care of. We perform adjustments depending on the type of node t. In cases (F1), (F3), and (F5) we do nothing. In case (F2) we add to $A(P)$ the unique arc in $\bar{B}_{TT}$ that covers t. Consider case (F4). If t is not covered by $\bar{B}_{SS}$ then we add to $A(P)$ the unique arc in F_S that leaves t.

Lemma 7 *The augmentation of $\bar{B}$ along P preserves properties* (P1)–(P3), (D1)–(D4), *and* (S1)–(S3). *The set of nodes in $V\bar{G}$ not covered by $\bar{B}$ strictly decreases. The set of initial nodes strictly decreases. The set of final nodes does not increase. The arc set of H decreases by AP.*

Proof Consider property (P1). We argue that arc sets $\bar{B}_{SS}$ and $\bar{B}_{TT}$ correspond to certain directed forests after the augmentation. This is achieved by an obvious case splitting according to the type of t. Cases (F1), (F3), (F5): sets $\bar{B}_{SS}$ and $\bar{B}_{TT}$ are not changed. Case (F2): set $\bar{B}_{SS}$ is not changed, set $\bar{B}_{TT}$ decreases by a single arc.

Case (F4): set $\bar{B}_{TT}$ is not changed, set $\bar{B}_{SS}$ may receive an additional arc from F_S. In all these cases set $\bar{B}_{TT}$ does not increase and also $\bar{B}_{SS} \subseteq AF_S$ holds after the augmentation (the latter also proves (S3)).

Consider property (P2). Clearly, it is sufficient to check only those arcs that are added to $\bar{B}$. Let a denote such an arc. If a is an $\bar{S}$–$\bar{T}$ arc then it is forward in P. Hence, $w_\pi(a) = 0$ holds by construction of H. Otherwise, a is a $\bar{S}$–$\bar{S}$ arc and case (F4) from the definition of a final node applies. Then, $a \in AF_S$ so (P2) follows from (S1).

Consider property (P3). For each node $v \in V\bar{G}$ put $c(v)$ (resp. $c'(v)$) to be the number of arcs in $\bar{B}$ that cover v before (resp. after) the augmentation along P. Clearly, $c'(v) > c(v)$ is only possible for an initial or final node v. More precisely, $c'(s) = 1$ holds, which satisfies (P3). Next, for the final node t one has $c'(t) > c(t)$ only in cases (F1) and (F3). In the former we have $\pi(\tilde{t}) = 0$, in the latter $c'(t) = 1$. Both are admissible.

Properties (D1)–(D4) are preserved since dual variables π are not changed. Properties (S1) and (S2) are preserved since neither π nor F_S is changed. Property (S3) is shown to hold by the above case splitting.

An easy case splitting shows that $c(v) \geq 1$ implies $c'(v) \geq 1$ for an arbitrary node $v \in V\bar{G}$. Also, $c(s) = 0$ and $c'(s) = 1$. Hence, the set of covered nodes increases by at least s.

The latter also implies that the set of initial nodes strictly decreases. As for the final nodes, consider a node $v \in V\bar{G}$ and perform the usual case splitting. Cases (F1), (F4): the status of v is not changed. Case (F2): v may only leave the set of final nodes (as $\bar{B}_{TT}$ cannot increase). Case (F3): similarly to the above (as the set of nodes covered by $\bar{B}$ may only increase). Case (F5): we claim that the augmentation cannot produce any new final nodes of this type. Indeed, for the contrary to happen one should have $c(v) \leq 1$ and $c'(v) \geq 2$. But c-value of a node in $\bar{S}$ may only increase for s and $c'(s) = 1$, which is a contradiction. Hence, the set of final nodes may only decrease.

Dealing with changes in AH is straightforward: construction of H implies that once an $\bar{S}$–$\bar{T}$ arc is added to or removed from $\bar{B}$, it vanishes from AH. All other $\bar{S}$–$\bar{T}$ arcs (not appearing in P) do not change their statuses. □

Note, that in order to achieve the desired time bound one cannot recompute path P from scratch each time. Taking into account the monotonicity of the sets of initial and final nodes, and the arc set of H, the *blocking augmentation* technique is applied (see, e.g., [3]). The latter computes, one by one, a sequence of augmenting paths and stops when there are no such paths left. Each of these paths is used for augmenting $\bar{B}$, as explained above. A more careful analysis will be given in Sect. 6.

A *dual step* is carried out when no more augmenting paths can be found. Each dual step consists of a sequence of dual updates. Let $\bar{S}_0$ (resp. $\bar{T}_0$) denote the set of nodes in $\bar{S}$ (resp. $\bar{T}$) that are reachable in H from initial nodes. We shall need the following easy but important observation:

Lemma 8 *At each dual step all nodes in $\bar{S}_0$ are roots of F_S.*

Proof If $v \in \bar{S}_0$ is an inner node of F_S then v is both reachable and final, namely, it has type (F4). This contradicts the absence of an augmenting path. □

Calculate the value of the *adjustment parameter* μ as follows:

$$\begin{aligned}
\mu_1 &:= \min\left(\pi(\tilde{v}) : v \in \bar{T}_0\right), \\
\mu_2 &:= \min\left(w_\pi(a) : a = (u, v) \in A\bar{G},\ u \in \bar{S}_0,\ v \in \bar{S}\right), \\
\mu_3 &:= \min\left(w_\pi(a) : a = (u, v) \in A\bar{G},\ u \in \bar{S}_0,\ v \in \bar{T} - \bar{T}_0\right), \\
\mu_4 &:= \min\left(1 - w_\pi(a) : a = (u, v) \in \bar{B},\ u \in \bar{S} - \bar{S}_0,\ v \in \bar{T}_0\right), \\
\mu &:= \min(\mu_1, \mu_2, \mu_3, \mu_4).
\end{aligned} \tag{4.1}$$

Clearly, (P2) implies that $\mu \geq 0$ and $\mu_4 \in \{0, 1, \infty\}$ (moreover, case $\mu_4 = 0$ is not possible, see Lemma 9 below).

The *dual update* is performed as follows. Firstly, put:

$$\begin{aligned}
\pi(\tilde{v}) &:= \pi(\tilde{v}) + \mu \quad \text{for each } v \in \bar{S}_0, \\
\pi(\tilde{v}) &:= \pi(\tilde{v}) - \mu \quad \text{for each } v \in \bar{T}_0.
\end{aligned}$$

If $w_\pi(a) = 0$ holds for some $a = (u, v) \in A\bar{G}$, $u \in \bar{S}_0$, $v \in \bar{S}$ after the update, then node u must be a root of F_S and property (S2) fails. Procedure NORMALIZE-FOREST is called to restore it. This might lead to a contraction inside $\bar{S}$.

Symmetrically, uncontractions in $\bar{T}$ are possible. The latter happen when the dual update decreases the dual $\pi(\tilde{v})$ of some complex node $v \in \bar{T}$ down to zero. This change violates (D4), so EXPAND is called to ensure it. (Note, that there might be multiple such calls per dual update since expansion of the top-most contracted subset $\tilde{v}$ may reveal additional contracted subsets with zero dual.)

After the dual update, sets $\bar{S}_0$ and $\bar{T}_0$ are incrementally recalculated, see Sect. 6. It will be shown later in Lemma 9 that these sets may only increase. This sequence of updates terminates when some final node becomes reachable, in which case a primal step is possible. The description of the dual step is now complete.

Lemma 9 *If no augmenting path exists in H then $0 < \mu < \infty$. The dual update preserves properties* (P1)–(P3), (D1)–(D4), *and* (S1)–(S3). *The set of initial nodes does not change. The set of final nodes does not decrease. Let R_0 denote the set of arcs in H that belong to some path from an initial node. Then, either the dual update leads to creation of a final node of type* (F2) *or does not decrease R_0.*

Proof First suppose that $\mu = 0$, hence one of $\mu_1, \ldots, \mu_4$ is zero. We proceed by an obvious case splitting.

1. If $\mu_1 = 0$ then $\pi(\tilde{v}) = 0$ holds for some node $v \in \bar{T}$. By (D4) the latter node is simple. Hence, v is a reachable final node of type (F1), which contradicts the absence of an augmenting path.
2. If $\mu_2 = 0$ then there is a reachable node $v \in \bar{S}_0$ that is covered by an $\bar{S}$–$\bar{S}$ arc a with $w_\pi(a) = 0$. By Lemma 8, all reachable nodes in $\bar{S}$ are roots of F_S. Hence, we get a contradiction with (S2).
3. If $\mu_3 = 0$ then there exists an $\bar{S}$–$\bar{T}$ arc $a = (u, v)$ such that $u \in \bar{S}_0$, $v \in \bar{T} - \bar{T}_0$, $w_\pi(a) = 0$. Two subcases are to be considered. Suppose $a \in \bar{B}$. Then either u is

reached by a (hence v is also reachable—a contradiction) or u is reached by some other $\bar{S}$–$\bar{T}$ arc belonging to $\bar{B}$ (hence u is a final node of type (F5), which is again a contradiction). On the other hand, if $a \notin \bar{B}$ then a is a forward arc in H, so v must be reachable, which is false.

4. Suppose $\mu_4 = 0$. There exists an arc $a = (u, v) \in \bar{B}$ such that $u \in \bar{S} - \bar{S}_0$, $v \in \bar{T}_0$, $w_\pi(a) = 1$. Then a is a backward arc in H and u is reachable.

In all these cases one has a contradiction proving that $\mu > 0$.

Now suppose that $\mu = \infty$ so each of the corresponding sets in (4.1) is empty. We argue that $\bar{G}$ has no bibranching, which is not possible since G has one. Indeed, $\mu_1 = \infty$ implies $\bar{T}_0 = \emptyset$. Consider an arbitrary initial node $s \in \bar{S}_0$ (there must be one, since otherwise the ST-stage is complete). It is not covered by an $\bar{S}$–$\bar{S}$ arc (for otherwise $\mu_2 < \infty$). Also, it is not covered by an $\bar{S}$–$\bar{T}$ arc. (otherwise its head must be in $\bar{T} = \bar{T} - \bar{T}_0$ and, therefore, $\mu_3 < \infty$). Hence, $\delta^{\text{out}}_{\bar{G}}(s) = \emptyset$ and $\bar{G}$ admits no bibranching.

Note that the dual update may perform contractions in $\bar{G}[\bar{S}]$ and uncontractions in $\bar{G}[\bar{T}]$ and, hence, change the node set of the current digraph $\bar{G}$. Therefore, some care is needed when proving the monotonicity of the sets of initial and final nodes.

Firstly, a node is initial if it belongs to $\bar{S}$ and is not covered by $\bar{B}$. The dual update does not change $\bar{B}$. When a subset $X \subseteq \bar{S}$ is contracted during NORMALIZE-FOREST call, it may contain at most one initial node, namely, the root node of the corresponding tree (by Lemma 8). The complex node generated by this contraction also becomes initial. In this sense, the set of initial nodes does not change.

Now consider the set of final nodes. An uncontraction may happen to a complex node $v \in \bar{T}_0$ when $\pi(\tilde{v})$ drops to zero. In this case $v \in \bar{T}_0$ so v cannot be final. Hence, uncontractions do not destroy existing final nodes. Additional final nodes of types (F1) and (F4), however, may appear. The former case (F1) happens when the dual $\pi(\tilde{v})$ becomes zero for some simple node $v \in \bar{T}_0$ or v is a complex node that, after expansion via the call to EXPAND, reveals some simple nodes with zero dual. The latter case (F4) is possible as NORMALIZE-FOREST may link a pair of trees and, hence, produce a new inner node in $\bar{S}$.

Next, we prove the required properties of R_0. Dealing with the change of π is simple. Let a be an arc belonging to a path P in H from an initial node. Then all nodes of P are reachable, hence the arcs of P correspond to $\bar{S}_0$–$\bar{T}_0$ arcs of $\bar{G}$. The reduced weights of these arcs are not affected, hence all arcs of P (in particular, a) remain present in H. Therefore, as long as no contractions or uncontractions are performed, R_0 may only increase. Contractions in $\bar{S}$ are also safe since they cannot destroy connectivity in H. Uncontractions in $\bar{T}$ are more tricky. Suppose that the dual $\pi(\tilde{v})$ associated to a reachable complex node $v \in \bar{T}$ drops to zero. EXPAND is applied to v thus replacing it with some set of nodes, say X'. Before the uncontraction, v was the head of at least one arc $a_1 = (u, v) \in AH$ such that u is also reachable. Moreover, there could be an arc $a_2 = (v, w) \in AH$. (Note that multiple arcs a_1 may exist in H but at most one arc a_2 is possible since it corresponds to an $\bar{S}$–$\bar{T}$ arc in $\bar{B}$ and the latter is unique by (P3).) After the uncontraction arcs a_1 and a_2 need not share a common endpoint (which was v). Hence, the uncontraction could potentially break the connectivity in H and, therefore, decrease set R_0. We argue that in this case the dual step completes since a final node of type (F2) is created. Indeed, let v'

be the tail of a_2 after the uncontraction. Now $\bar{B}$ is extended by adding the arc set of a directed out-tree that covers X' and is rooted at v'. If a_1 ends in v' (after the uncontraction) then a_1 and a_2 still share an endpoint and the connectivity in H is preserved. Otherwise, a_1 ends in some inner node of $\bar{B}_{TT}$ and, hence, its head is a reachable final node of type (F2).

Validity of properties (D1) and (D2) is clear. Property (D3) is preserved by the choice of μ and the structure of the dual adjustment. Property (D4) is ensured by the calls to EXPAND.

Consider property (S1). By Lemma 8 no inner node of F_S is a member of $\bar{S}_0$. For the subgraph $G[\bar{S}]$, the dual step only changes the reduced weights of arcs leaving nodes in $\bar{S}_0$. Therefore, no arc in AF_S changes its reduced weight. Property (S2) follows since the dual updates make calls to NORMALIZE-FOREST. Also, such a call is the only way of changing $\bar{B}$ and F_S during the dual step. Hence, property (P1) remains valid. Property (S3) is trivially preserved.

Consider property (P2). Since $\bar{B}_{SS} \subseteq AF_S$ and property (S1) is preserved, $w_\pi(a) = 0$ holds for all $a \in \bar{B}_{SS}$. The dual update may only decrease the reduced weight of an arc $a = (u, v) \in \bar{B}_{TT}$ if $u \in \bar{T} - \bar{T}_0$ and $v \in \bar{T}_0$. But existence of a would imply that v is a reachable final node of type (F2), which is a contradiction. Hence, the reduced weights of $\bar{T}$–$\bar{T}$ arcs in $\bar{B}$ remain zero. Finally, consider an arc $a = (u, v) \in \bar{B}_{ST}$ and suppose that the dual update increases $w_\pi(a)$. The latter is only possible if $u \in \bar{S} - \bar{S}_0$ and $v \in \bar{T}_0$. Therefore, $\mu_4 \leq 1$ by (4.1). Since $\mu > 0$ this implies $\mu = \mu_4 = 1$ and hence $w_\pi(a)$ increases from 0 to 1, which is admissible.

Consider property (P3). Adjustments (4.2) only increase duals for subsets corresponding to nodes in $\bar{S}$. Let $v \in \bar{S}$ be such a node. Moreover, suppose that v is covered by at least two arcs in $\bar{B}$. If $v \in S_0$, then v is a root of F_S (by Lemma 8). Therefore, by (S3) node v is covered by at least two $\bar{S}$–$\bar{T}$ arcs in $\bar{B}$ and, hence, v is a final node of type (F5), which is a contradiction with the absence of an augmenting path. So one has $v \in \bar{S} - \bar{S}_0$ and the dual update preserves the corresponding dual $\pi(\tilde{v})$. □

If an augmenting path arises after these changes, the dual step completes and a primal step is executed. Otherwise, the next value of μ is calculated and the process of changing π proceeds.

The algorithm changes $\bar{B}$, π, and $\mathcal{F}$ by executing primal and dual steps alternatingly. It stops when $\bar{B}$ covers all nodes in $\bar{S}$. Then, the parts $\bar{S}$ and $\bar{T}$ are exchanged and the TS-stage runs until $\bar{B}$ covers both $\bar{S}$ and $\bar{T}$. This way, the requested bibranching $\bar{B}$ is constructed, the scaling step completes.

5 Complexity Analysis

In this section we present a high-level efficiency analysis of the algorithm. Our immediate goal is to prove an $O(\sqrt{n})$ bound for the number of primal and dual steps during each scaling step. Hereinafter w, π_0, $\mathcal{F}_0$, and $\bar{G}_0$ denote the corresponding objects after the doubling stage. Similarly, we use notation π_1, $\mathcal{F}_1$, and $\bar{G}_1$ when referring to the state immediately after the shell stage.

Lemma 10 *There exists a basic π_1-consistent bibranching B_1 in G obeying $w_{\pi_1}(B_1) \le 6n$.*

Proof Let $\bar{B}_0$ be a bibranching in $\bar{G}_0$ that was constructed by the previous scaling step. In case of the first scaling step one has $G_0 = G$; we put $\bar{B}_0$ to be an arbitrary bibranching in G. By removing an appropriate set of arcs from $\bar{B}_0$ one may assume that $\bar{B}_0$ is basic, see Lemma 2. Property (P2) and the structure of the doubling stage imply that $w_{\pi_0}(a) \le 3$ holds for each $a \in \bar{B}_0$.

We now gradually transform the digraph $\bar{G}_0$ into $\bar{G}_1$ and, simultaneously, $\bar{B}_0$ into a bibranching $\bar{B}_1$ in $\bar{G}_1$ such that $w_{\pi_1}(\bar{B}_1)$ is small and (P3) holds for $\bar{B} := \bar{B}_1$, $\pi := \pi_1$. Note that this construction of $\bar{B}_1$ is independent from the algorithm.

Let us denote the current digraph by $\bar{G}$ and the current bibranching by $\bar{B}$. Initially, $\bar{G} := \bar{G}_0$ and $\bar{B} := \bar{B}_0$. The difference between $\bar{G}_0$ and $\bar{G}_1$ is that some maximal sets in $\mathrm{supp}(\pi_0)$ may dissolve during the shell stage. The corresponding dissolved nodes, therefore, are replaced by certain subgraphs.

We enumerate the nodes of $\bar{G}_0$, let v be the current one. If $\tilde{v} \notin \mathrm{supp}(\pi_0)$ then v is a simple node ($\tilde{v} = \{v\}$), it remains simple in $\bar{G}_1$. No change is applied to $\bar{G}$ and $\bar{B}$. Note that the reduced weights of arcs covering v are not changed during the shell stage and, thus, do not exceed 3.

Next, suppose $\tilde{v} \in \mathrm{supp}(\pi_0)$. We assume that $\tilde{v} \subseteq S$ (the other case is symmetric). Property (P3) implies that in $\bar{G}_0$ node v is covered by a unique arc, say $a_0 \in \bar{B}_0$. Let the tail of a_0 in G be w. Applying (D2) iteratively, we construct an in-tree W in G such that: (i) W is rooted at w; (ii) $VW = \tilde{v}$; (iii) arc set $A := AW \cup \{a_0\}$ is π_0-consistent; (iv) every arc $a \in AW$ was of zero reduced weight prior to the doubling stage. Clearly, $w_{\pi_0}(a) \le 3$ holds for every $a \in A$.

Update $\bar{G}$ and $\bar{B}$ as follows. First, uncontract $\tilde{v}$ completely and add AW to $\bar{B}$. Since node w is reachable from every node in $\tilde{v}$ by arcs in AW, it follows that $\bar{B}$ remains a bibranching. Next, contract the maximal sets $X \in \mathrm{supp}(\pi_1)$ such that $X \subseteq \tilde{v}$ and update $\bar{B}$ accordingly. Let $\bar{v}$ denote the image of $\tilde{v}$ under these contractions. (It is possible that the whole set $\tilde{v}$ gets contracted again, this happens when $\tilde{v} \in \mathrm{supp}(\pi_1)$; in this case $\tilde{v}$ did not dissolve during the call NORMALIZE-SHELL($\tilde{v}$).)

The above contractions may remove some arcs from $\bar{B}$ (more precisely, exactly those arcs whose head and tail nodes are simultaneously contained in the same maximal contracted set). However, $\bar{B}$ remains a bibranching since contraction of an arbitrary subset of S- or T-part of the digraph preserves the required connectivity. Finally, we apply Lemma 2 and remove all redundant arcs from $\bar{B}$ (in an arbitrary way) turning it into a basic bibranching.

Recall that initially v was covered by the unique arc a_0. Now v is expanded into some set of nodes, namely $\bar{v}$, and some arcs from $\gamma(\tilde{v})$ are added to $\bar{B}$. Since $\bar{B}$ is basic, every node in $\bar{v}$ is covered by a unique arc from $\bar{B}$. This way, (P3) follows for $\bar{B}$.

Let us estimate the total reduced weight w_{π_1} of all newly added arcs in $\bar{B}$ (including arc a_0). To this aim, we bound $w_{\pi_1}(A)$ (since $\bar{B}$ receives some subset of arcs from A and reduced weights of the omitted arcs are non-negative). We consider the following two subfamilies of $\mathrm{supp}(\pi_0)$ and $\mathrm{supp}(\pi_1)$:

$$\mathcal{F}_0 := \{X \in \mathrm{supp}(\pi_0) \mid X \subseteq \tilde{v}\}, \qquad \mathcal{F}_1 := \{X \in \mathrm{supp}(\pi_1) \mid X \subseteq \tilde{v}\}.$$

During NORMALIZE-SHELL($\tilde{v}$), each time the dual variable corresponding to a set $X \subseteq \tilde{v}$ is increased by μ, the dual variable corresponding to some other set $Y \subseteq \tilde{v}$ is decreased by the same value μ. Hence,

$$\sum(\pi_0(X) : X \in \mathcal{F}_0) = \sum(\pi_1(X) : X \in \mathcal{F}_1).$$

Recall that A is π_0-consistent. Hence, by Lemma 3 it follows that

$$w_{\pi_0}(A) = w(A) - \vartheta\pi_0(A) = w(A) - \sum(\pi_0(X) : X \in \mathcal{F}_0),$$

$$w_{\pi_1}(A) = w(A) - \vartheta\pi_1(A) \le w(A) - \sum(\pi_1(X) : X \in \mathcal{F}_1).$$

Therefore,

$$w_{\pi_1}(A) \le w_{\pi_0}(A) \le 3|A| = 3|\tilde{v}|.$$

The above procedure is applied to each node $v \in V\bar{G}_0$ and eventually stops with $\bar{G} = \bar{G}_1$. The final set $\bar{B}$ is denoted by $\bar{B}_1$. Let us estimate its reduced weight $w_{\pi_1}(\bar{B}_1)$. First, $\bar{B}_1$ gets at most n arcs that cover simple nodes in $\bar{G}_0$; each of those arcs has a reduced weight not exceeding 3. Second, each complex node $v \in V\bar{G}_0$ generates a set of arcs with total reduced weight not exceeding $3|\tilde{v}|$. Summing these bounds, one gets:

$$w_{\pi_1}(\bar{B}_1) \le 3n + 3n = 6n.$$

Finally, to get the desired bibranching B_1 in G we apply EXPAND routine and extend $\bar{B}_1$ into the maximal contracted sets of $\bar{G}_1$. This step only adds arcs of zero reduced weight w_{π_1}. Hence, $w_{\pi_1}(B_1) = w_{\pi_1}(\bar{B}_1) \le 6n$ holds. □

Lemma 11 *Let $\pi : 2^S \cup 2^T \to \mathbb{R}_+$ be an arbitrary feasible dual solution (that is, w_π is non-negative) and $B \subseteq AG$ be an arbitrary π-consistent arc set. Define*

$$\Delta(\pi, \pi_1, B) := \sum\big(\pi(X) - \pi_1(X) : X \text{ is not covered by } B\big).$$

Then $\Delta(\pi, \pi_1, B) \le 6n + w_\pi(B)$.

Proof Consider the duals π_1 (at the moment just before the ST-stage) and a basic π_1-consistent bibranching B_1 constructed in Lemma 10. Put

$$Q := \sum_{a \in AG} \big(\chi^{B_1}(a) - \chi^{B}(a)\big)\big(w_{\pi_1}(a) - w_\pi(a)\big).$$

(Here χ^U denotes the *incidence vector* of an arc set U, i.e. the function that equals 1 on U and 0 on $AG - U$.)

Taking equalities $w_\pi = w - \vartheta\pi$ and $w_{\pi_1} = w - \vartheta\pi_1$ into account one gets

$$\begin{aligned} Q &= \sum_{a \in AG} \big(\chi^{B_1}(a) - \chi^{B}(a)\big)\big(\vartheta\pi(a) - \vartheta\pi_1(a)\big) \\ &= \vartheta\pi(B_1) + \vartheta\pi_1(B) - \vartheta\pi_1(B_1) - \vartheta\pi(B). \end{aligned}$$

Since B is π-consistent and B_1 is π_1-consistent and covers each set in $2^S \cup 2^T$, Lemma 3 implies

$$\begin{aligned} Q &\geq \sum\bigl(\pi(X) : X \text{ is covered by } B_1\bigr) + \sum\bigl(\pi_1(X) : X \text{ is covered by } B\bigr) \\ &\quad - \sum\bigl(\pi_1(X) : X \text{ is covered by } B_1\bigr) - \sum\bigl(\pi(X) : X \text{ is covered by } B\bigr) \\ &= \sum\bigl(\pi(X) : X \text{ is not covered by } B\bigr) - \sum\bigl(\pi_1(X) : X \text{ is not covered by } B\bigr) \\ &= \sum\bigl(\pi(X) - \pi_1(X) : X \text{ is not covered by } B\bigr) = \Delta(\pi, \pi_1, B). \end{aligned}$$

On the other hand, since w_π and w_{π_1} are non-negative

$$\begin{aligned} Q &= \sum_{a \in AG} \bigl(\chi^{B_1}(a) - \chi^{B}(a)\bigr)\bigl(w_{\pi_1}(a) - w_\pi(a)\bigr) \\ &\leq \sum_{a \in AG} \chi^{B_1}(a) w_{\pi_1}(a) + \chi^{B}(a) w_\pi(a) \\ &= w_{\pi_1}(B_1) + w_\pi(B) \leq 6n + w_\pi(B). \end{aligned}$$

Now the claim follows by transitivity. □

Lemma 12 *Each scaling step executes $O(\sqrt{n})$ primal and dual steps. Totally, these dual steps execute $O(n)$ dual updates.*

Proof Let $\bar{B} \subseteq A\bar{G}$ denote the current partial bibranching in the current digraph $\bar{G}$ and π be the current duals at some intermediate moment during an ST-or a TS-stage.

Firstly, we prove that $w_\pi(\bar{B}) \leq n$. Indeed $w_\pi(\bar{B})$ does not exceed the number of $\bar{S}$–$\bar{T}$ arcs in $\bar{B}$ (by property (P2)). The latter may only increase by 1 on each primal step. The total number of primal steps does not exceed n (since each of these steps decreases the set of uncovered nodes).

Next, similarly to Lemma 4 we apply EXPAND routine, and extend $\bar{B}$ to a π-consistent set $B \subseteq AG$. The latter obeys $w_\pi(B) = w_\pi(\bar{B}) \leq n$.

Consider a set $X \subseteq VG$ and suppose that $\pi(X) < \pi_1(X)$. It follows from the structure of the algorithm that B covers X. Indeed, when the dual $\pi(\tilde{v})$ decreases (for $v \in V\bar{G}$), $\bar{B}$ covers v for otherwise v is a reachable final node of type (F3) and the dual adjustment is not possible. Also, if $\bar{B}$ covers v then B covers $\tilde{v}$ and all its subsets. This proves that all the terms in $\Delta(\pi, \pi_1, B)$ are non-negative.

Consider the ST-stage. Let k be the number of dual steps performed so far. Initially, all nodes in $\bar{S}$ are not covered by $\bar{B}$. Then, during the course of the algorithm the number of uncovered nodes in $\bar{S}$ decreases. For each i $(1 \leq i \leq k)$ let $u^i_1, \ldots, u^i_{l_i}$ denote the uncovered nodes in $\bar{S}$ during the i-th dual step.

For each fixed i, the sets $\tilde{u}^i_1, \ldots, \tilde{u}^i_{l_i}$ are pairwise disjoint. Moreover, for each set $\tilde{u}^i_j$, $i < k$, $1 \leq j \leq l_i$, there are exactly two possibilities: (i) node u^i_j gets covered during the upcoming primal step and, hence, $\tilde{u}^{i'}_{j'} \cap \tilde{u}^i_j = \emptyset$ for each $i' > i$ and $1 \leq$

$j' \le l_{i'}$; or (ii) node u^i_j gets incorporated into some contracted set at the next dual step, hence, $\tilde{u}^i_j \subseteq \tilde{u}^{i+1}_{j'}$ for some $1 \le j' \le l_{i+1}$.

Therefore, we may associate a forest F with the above sets as follows. Nodes of F are pairs (i, j), where $1 \le i \le k$, $1 \le j \le l_i$ (hence, nodes of F correspond to the uncovered nodes in $\bar{S}$ at different moments of time during the course of the execution). Pair (i, j) is an immediate descendant of $(i+1, j')$ in F if $\tilde{u}^i_j \subseteq \tilde{u}^{i+1}_{j'}$.

Let $H_1, \ldots, H_{l_k}$ be the trees of F (rooted at pairs $(k, 1), \ldots, (k, l_k)$). All these trees are of height k. We argue that for each tree H_j and each level i $(1 \le i \le k)$ there exists a pair (i, j) in tree H_j such that B does not cover $\tilde{u}^i_j$. This follows from the way B is constructed from $\bar{B}$. Indeed, let EXPAND be applied to uncontract a set $X \subseteq VG$ and update the current partial bibranching (call it A). Additionally, suppose A does not cover X. After uncontraction X gives rise to a set of nodes X' (following notation from Sect. 3). Then, the updated partial bibranching A' covers all nodes in X' except one.

Fix a tree H_j and a level i $(1 \le i \le k)$. Let j the be the index corresponding to an uncovered set $\tilde{u}^i_j$, as constructed above. Node u^i_j was initial (at the corresponding moment of time) and, hence, reachable. Therefore, $\pi(\tilde{u}^i_j)$ was increased during the corresponding dual step. Pair (i, j) contributes the term $\pi(X) - \pi_1(X)$ for $X := \tilde{u}^i_j$ to $\Delta(\pi, \pi_1, B)$. Summing over all levels i (but keeping H_j fixed) one concludes that H_j contributes $\sum \mu$ to $\Delta(\pi, \pi_1, B)$, where $\sum \mu$ is the sum of the dual adjustments μ performed by the algorithm during the ST-stage.

Now summing over all the trees $H_1, \ldots, H_{l_k}$ one gets

$$\Delta(\pi, \pi_1, B) \ge \sum \mu \cdot l_k. \tag{5.1}$$

On the other hand, by Lemma 11

$$\Delta(\pi, \pi_1, B) \le 6n + w_\pi(B) \le 7n. \tag{5.2}$$

Therefore, $\sum \mu \cdot l_k \le 7n$. This implies $\sum \mu = O(n)$ and, hence, as each dual update changes the duals by at least 1, the total number of the latter is $O(n)$, as claimed. Moreover, each dual step executes at least one dual update, hence $k \cdot l_k \le 7n$. After $\lceil \sqrt{n} \rceil$ dual steps at most $O(\sqrt{n})$ nodes in $\bar{S}$ remain uncovered. To cover these remaining nodes $O(\sqrt{n})$ primal steps are sufficient (as each such step decreases the number of uncovered nodes by 1). Primal and dual steps are executed alternatingly, hence, the bound of $O(\sqrt{n})$ holds for both of them.

Next, consider the state after k dual steps in the TS-stage and let, as earlier, π and $\bar{B} \subseteq A\bar{G}$ denote the current duals and the current partial bibranching, respectively. Put $B \subseteq AG$ to be the result of expanding $\bar{B}$. We have shown earlier that there are no negative terms in $\Delta(\pi, \pi_1, B)$. Moreover, the very same argument (with $\bar{S}$ and $\bar{T}$ exchanged) applies, so one may consider subsets $\tilde{u}^i_j \subseteq T$ similarly to the ST-stage. Since each of these sets $\tilde{u}^i_j$ corresponds to an uncovered node in $\bar{T}$ at some intermediate moment and the algorithm may only decrease duals of sets that are covered, it follows that none of $\pi(\tilde{u}^i_j)$ is changed during the ST-stage. Thus, (5.1) and (5.2) hold for the TS-stage as well, and the latter completes after executing $O(\sqrt{n})$ primal and dual steps and $O(n)$ dual updates. □

Applying ideas from [4, 6, 9, 13] one can implement the shell stage, the primal, and the dual steps to run in $O(m \log n)$ time each. The necessary details are given in the upcoming Sect. 6. Since the total number of scaling steps is $(\lfloor \log W \rfloor + 1) + (\lfloor \log n \rfloor + 1) = O(\log(nW))$ (see Sect. 3) we conclude as follows:

Theorem 2 *The running time of the algorithm is* $O(m\sqrt{n}\, \log n \log(nW))$.

6 Implementation Details

In this section we discuss the details of an $O(m\sqrt{n}\, \log n \log(nW))$-time implementation. In particular, we indicate how the shell stage, the primal, and the dual steps can be carried out in $O(m \log n)$ time each.

Let us start with the shell stage. Family $\mathcal{F}$ gives rise to a *contraction forest* that is denoted by $\hat{\mathcal{F}}$. The leafs of $\hat{\mathcal{F}}$ correspond to (and are identified with) the nodes in G. All other (inner) nodes of $\hat{\mathcal{F}}$ correspond to (and are identified with) the elements of $\mathcal{F}$. The "child–parent" relation in $\hat{\mathcal{F}}$ is defined in a natural way. To make contractions and uncontractions efficient, digraph $\bar{G}$ and $S(X)$ for $X \in \mathcal{F}$ are never maintained explicitly. Instead, contraction forest essentially encodes the difference between G and these graphs. The values $\pi(X)$, $X \in \mathcal{F}$, are attached directly to the nodes of $\hat{\mathcal{F}}$. For a non-leaf node $X \in V\hat{\mathcal{F}}$ the children of X in $\hat{\mathcal{F}}$ are regarded as nodes of $S(X)$. To make EXPAND efficient, we store the arc set of the corresponding Hamiltonian cycle in $S^0_\pi(X)$.

Dealing with arcs is slightly more difficult. One can easily see that the S–T arcs may be ignored for the duration of the shell stage. We call a node $X \in V\hat{\mathcal{F}}$ *active* if either X is a root node in $\hat{\mathcal{F}}$ or X has an unscanned ancestor Y in $\hat{\mathcal{F}}$. For each active node X we maintain a list $A(X)$ of certain arcs in G. This list contains all S–S or T–T arcs $a \in AG$ such that the image of a is present in $S(Y)$ (if Y is the ancestor of X) or $\bar{G}$ (if X is a root node) and covers X. Also, $A(X)$ may contain additional arcs from $\gamma_G(X)$. The latter arcs are called *dead*, the algorithm drops them once they are discovered, see below.

These lists are constructed as follows. Given an S–S or T–T arc $a = (u, v) \in AG$, $u, v \in VG$, consider the leafs u and v in $\hat{\mathcal{F}}$. If u and v are in different contraction trees then a appears in $\bar{G}$ only. We add a to $A(U)$ (if a is an S–S arc) or $A(V)$ (if a is a T–T arc). Here U (resp. V) is the root node of the subtree containing u (resp. v). Otherwise (u and v are in the same tree), let W be the least common ancestor (LCA) of u and v in $\hat{\mathcal{F}}$. Then, for an S–S arc a it is attached to the child of W such that u is its descendant. Similarly, if a is a T–T arc then we attach a to the child of W such that v is its descendant. Plenty of methods are known to solve the above LCA problem, we may apply any of them that requires $O(\log n)$ time per query.

For consistency, we regard the leafs of $\hat{\mathcal{F}}$ as scanned regardless the fact that the corresponding singleton sets never appear in $\mathcal{F}$. Hence, the set of scanned nodes in $\hat{\mathcal{F}}$ is closed under taking descendants. Let $\hat{\mathcal{F}}_0$ denote the subforest of $\hat{\mathcal{F}}$ consisting of all scanned nodes. Hence, at the beginning of the shell stage $\hat{\mathcal{F}}_0$ only contains leafs that correspond to the nodes of G. During the course of the shell stage other scanned nodes may appear in $\hat{\mathcal{F}}$; these nodes are added to $\hat{\mathcal{F}}_0$. We use a forest of *dynamic*

trees [16] here. For each leaf $x \in V\hat{\mathcal{F}}_0$, the root node $r(x)$ of the corresponding tree in $\hat{\mathcal{F}}_0$ and the value $\sigma(x)$ that is the sum of duals $\pi(Y)$ for all nodes Y on the path from x to $r(x)$ in $\hat{\mathcal{F}}_0$ can be computed in $O(\log n)$ time per request. Also, updating a single dual attached to a node of $\hat{\mathcal{F}}_0$, adding an edge (or a node) to $\hat{\mathcal{F}}_0$ or removing an edge (or a node) from $\hat{\mathcal{F}}_0$ costs $O(\log n)$ amortized time. Totally, maintenance of $\hat{\mathcal{F}}_0$ takes $O(n \log n)$ time for the whole shell stage (only $O(n)$ contractions and uncontractions are possible, only $O(n)$ edges and nodes are totally added to $\hat{\mathcal{F}}_0$ and removed from $\hat{\mathcal{F}}_0$).

These lists $A(X)$ enable the algorithm to access the appropriate set of outcoming (for $X \subseteq S$) or incoming (for $X \subseteq T$) arcs at any point with no additional overhead (except that caused by the presence of dead arcs). Next we explain how the algorithm deals with reduced arc weights and computes the values of μ. To this aim, consider an invocation NORMALIZE-SHELL(X), where X denotes the current node of $\hat{\mathcal{F}}$ to be scanned. Set X is still unscanned but all its descendants are already scanned. According to our definition the children of X in $\hat{\mathcal{F}}$ are identified with the nodes of $S(X)$; these are some active nodes. For each such child Y the algorithm turns $A(Y)$ into a meldable priority queue, e.g. into a *leftist heap* (see, e.g., [17]). Each heap insertion costs $O(\log n)$ time units. We use reduced weights as keys in these heaps. The latter are calculated as follows. Consider an arc $a = (u, v) \in A(Y)$. The algorithm takes the initial value of $w(a)$ and then adjusts it by subtracting $\sigma(u)$ (for the case $X \subseteq S$) or $\sigma(v)$ (for the case $X \subseteq T$). (One must recall here that S–T arcs are ignored and may not appear in $A(Y)$.)

To compute the value of μ during a dual adjustment, the algorithm fetches an arc $a = (u, v) \in AG$ with the minimum reduced weight from the appropriate heap. If this arc is dead (which can be detected in $O(\log n)$ time by making a query to the forest of dynamic trees and comparing $r(u)$ and $r(v)$) it is discarded and another attempt is made. Fetching an arc with the minimum key costs $O(\log n)$ time.

One can easily adjust the implementation of the leftist heap in such a way that is supports decreasing all its keys in $O(1)$ time. Instead of storing the actual keys in nodes of a leftist heap we store certain "delta values" there. The actual key that corresponds to a node v is the sum of deltas on the path from the root node to v. Changing all keys in a heap can be carried out by changing the delta value assigned to its root. Also, these deltas may be handled by the standard heap melding operation (see [17]) with no additional overhead.

Recall that the shell state grows a certain forest in the current shell digraph $S(X)$. When arc a is picked, two cases are possible. Namely: (i) either arc a connects a pair of distinct trees; or (ii) arc a forms a cycle. To distinguish between these cases a disjoint set union (DSU) data structure [2] is employed. It keeps the partition of $VS(X)$ into the node sets of trees of the said forest. The total time to maintain DSU is $O(m \log n)$ (assuming any reasonable implementation).

Dealing with the case (i) is straightforward. In case (ii) the set of nodes of $S(X)$ to be contracted is easily constructed in time proportional to its size. Forests $\hat{\mathcal{F}}$ and $\hat{\mathcal{F}}_0$ are adjusted to reflect the creation of a new active set, say $Y \subseteq VG$. All children of Y are already scanned. Node $Y \in V\hat{\mathcal{F}}$ is also marked as scanned. To construct $A(Y)$ one melds the heaps of the children of Y (these heaps are no longer needed since they correspond to inactive nodes). Note that this is exactly the point where a

dead arc may appear in a list. Altogether, heap management during contractions take $O(n \log n)$ time for the whole shell stage.

NORMALIZE-SHELL(X) may stop for two reasons. It either restores property (D2) for set X or decreases $\pi(X)$ down to zero and, hence, dissolves X. Consider the latter case. Set X is removed from $\hat{\mathcal{F}}$, so all of its children are attached to the ancestor of X (if any). Also, the algorithm scans the arcs in $A(X)$ and attaches them to the appropriate children of X. Namely, let $a = (u, v)$ be an arc in $A(X)$. Then $r(u)$ (for $X \subseteq S$) or $r(v)$ (for $X \subseteq T$) is exactly the node of $\hat{\mathcal{F}}$ where a belongs. Any arc in G may be scanned this way at most once. Also each arc in G is fetched from the appropriate heap during the shell stage at most once. Hence, each shell stage totally takes $O(n \log n + m \log n) = O(m \log n)$ time, as claimed.

Next, consider the ST-stage (the TS-stage is similar). Now the S–T arcs of G must be taken into account. Forest $\hat{\mathcal{F}}$, DSU for F_S are maintained as a forest of dynamic trees similarly to the shell stage. In particular, values $r(v)$ and $\sigma(v)$, $v \in VG$, can be computed in $O(\log n)$ time per request. Note that these values enable us to check if an arc $a \in AG$ is present in $\bar{G}$ and also to compute its endpoints in $\bar{G}$ and the reduced weight in $O(\log n)$ time.

Let us first deal with the S-part of the G. For each root node $X \in V\mathcal{F}$ such that $X \subseteq S$ we maintain two lists. List $A_S(X)$ (resp. $A_T(X)$) contains all S–S (resp. S–T) arcs in AG whose images leave X in $\bar{G}$. Also, these lists may contain dead arcs (those belonging to $\gamma_G(X)$). When a subset of nodes in $\bar{S}$ gets contracted in $\bar{G}$, a new root node, say Y, is formed in $\hat{\mathcal{F}}$; its lists $A_S(Y)$ and $A_T(Y)$ are constructed by merging the corresponding lists of its children in $\hat{\mathcal{F}}$ (in time proportional to the number of lists to be merged). Similar to the shell stage, these merges may produce dead arcs, which are discarded once fetched; this incurs $O(m \log n)$ of total time overhead (each arc costs $O(\log n)$ and may be discarded at most once).

We now explain how the auxiliary digraph H is constructed and maintained during the ST-stage. As usual, the node set of H just coincides with the set of roots of $\hat{\mathcal{F}}$. To check if a node in H is initial and to distinguish cases (F2), (F3), and (F5) in the definition of a final node the algorithm maintains, for each node v in $\bar{T}$ and each inner (w.r.t. forest F_S) node v in $\bar{S}$, a counter indicating the number of arcs in $\bar{B}$ that cover v. Checking for cases (F1) and (F4) is straightforward.

Consider a primal step. To construct H we scan the lists $A_T(v)$ for all $v \in \bar{S}$ and choose the arcs with zero reduced weight. Also, we scan the current partial bibranching $\bar{B}$ and add the appropriate backward arcs to H. Altogether this takes $O(m \log n)$ time ($O(\log n)$ time per arc). At each primal step the blocking path method [3] is applied to H. We construct, in $O(m)$ total time, the required set of augmenting paths, one path at a time. Augmenting $\bar{B}$ along each of these paths P in $O(|AP|)$ time is easy. Hence, each primal step is carried out in $O(m \log n)$ time.

Finally, we consider a dual step. One can maintain sets $\bar{S}_0$ and $\bar{T}_0$ in H incrementally as follows. At the beginning of the dual step the algorithm constructs H and computes the sets $\bar{S}_0$ and $\bar{T}_0$ explicitly in $O(m)$ time. When a new arc (u, v) appears in H the algorithm checks if u is known to be reachable. In the latter case node v is declared reachable as well (i.e. added either to $\bar{S}_0$ or $\bar{T}_0$). Processing each new arc in H takes $O(1)$ time, therefore the whole maintenance of $\bar{S}_0$ and $\bar{T}_0$ costs $O(m)$ for each dual step.

However, there are complications that are caused by contractions and uncontractions. These operations are applied to $\bar{G}$ and hence can also change the node set of H. We need to take them into account and update $\bar{S}_0$ and $\bar{T}_0$ appropriately.

Dealing with a contraction of a subset $X \subseteq \bar{S}$ is fairly easy. As explained earlier, we update the DSU corresponding to F_S. The newly created complex node is reachable iff at least one node in X was reachable before the contraction.

Dealing with uncontractions in $\bar{T}$ is more tricky. Let EXPAND be called for a reachable complex node $v \in \bar{T}$ and replaces it with a set of nodes X'. One has to decide which nodes in X' are reachable. Consider the arc set R_0 as in the statement of Lemma 9. Let R_1 be the set of S–T arcs than generate forward arcs in R_0. The members of R_1 are arcs in H, however they may be regarded as arcs in G. Lemma 9 indicates that R_1 only increases (with a possible exception of the last dual update that calls EXPAND and creates a final node of type (F2)). The algorithm maintains R_1, which takes $O(m)$ time for the whole dual step. Also, for each node $u \in T$ the algorithm keeps the number of arcs in R_1 entering u (recall that we regard elements of R_1 as arcs in G). Now a node $u \in \bar{T}$ is reachable if and only if there exists a node $u' \in \tilde{u}$ such that u' has a positive counter of R_1-arcs. These R_1-counters can be maintained and for each u existence of u' can be checked in $O(\log n)$ time by a simple augmentation of $\hat{\mathcal{F}}$.

Therefore, we process the uncontraction of v as follows. Firstly, we remove v from $\bar{T}_0$ (since v is no longer present in $\bar{G}$ and H). Next, the elements of X' are scanned and reachable nodes among them are selected and added to $\bar{T}_0$, as explained above. If a final node of type (F2) is discovered, the dual step completes. (In this case we do not know the actual set of reachable nodes as the connectivity in H could get broken, see the proof of Lemma 9. However, this is unimportant since the upcoming primal step will compute H from scratch anyway.) Otherwise, the set of reachable nodes is correctly updated.

Processing each new arc in H and updating $\bar{S}_0$ and $\bar{T}_0$ costs $O(1)$ time. Processing each new arc in R_1 costs $O(\log n)$ amortized time (as it involves updating the counters in $\hat{\mathcal{F}}$). Dealing with contractions and uncontractions takes $O(n \log n)$ time. Therefore, $O(m \log n + n \log n) = O(m \log n)$ time is sufficient to maintain $\bar{S}_0$ and $\bar{T}_0$ during each dual step.

During the dual step the potentials of nodes in $\bar{S}_0$ are increased by μ and the potentials of nodes in $\bar{T}_0$ are decreased by μ, see (4.2). A trivial modification to the bookkeeping of duals allows making these changes in $O(1)$. The bottleneck of the dual step comes from the maintenance of sets $\bar{S}_0$, $\bar{T}_0$ and computation of the appropriate adjustment parameters μ, see (4.1).

First, a priority queue Q_1 is organized that stores the duals corresponding to the nodes in $\bar{T}_0$. Computing a single value of μ_1 takes $O(\log n)$ time. By Lemma 12 each dual step executes $O(n)$ dual updates, hence each dual step takes $O(n \log n)$ time for computing μ_1 and maintaining Q_1.

Second, lists $A_S(X)$ are turned into leftist heaps (reduced arcs weights are regarded as keys). A second-order priority queue Q_2 is organized that captures, for each node $v \in \bar{S}_0$, the minimum from the corresponding heap $A_S(v)$. The total time for computing μ_2 and maintaining Q_2 is again $O(n \log n)$. If the adjustment produces an arc $a \in A_S(X)$ with zero reduced weight this arc is extracted and examined. If a is

dead, it is skipped. Otherwise, (S2) property fails and NORMALIZE-FOREST routine applies as described in Sect. 4.

Third, let us focus on computing μ_3. To this aim, the algorithm maintains a priority queue Q_3 that contains the S–T arcs (u, v) such that $u \in \bar{S}_0$ and $v \in \bar{T} - \bar{T}_0$ and, possibly, some other arcs. Queue Q_3 is incrementally constructed by scanning the nodes in $u \in \bar{S}_0$ and examining their lists $A_T(u)$. Each arc that is added to Q_3 takes $O(\log n)$ time for computing its reduced weight. Also, as the algorithm progresses, more nodes get added to $\bar{T}_0$, hence certain previously added arcs no longer enter $\bar{T} - \bar{T}_0$. These arcs are considered *dead*; a usual lazy cleanup is used. As earlier, the algorithm regards reduced weights as keys in Q_3. An arc may be added to Q_3 and a minimum may be extracted from Q_3 in $O(\log n)$ time. Also, all keys in Q_3 may be decreased by μ in $O(1)$ time (using the earlier mentioned delta encoding for storing keys). Hence, working with Q_3 during each dual step takes $O(m \log n)$ time.

Fourth, consider μ_4. To compute it in $O(1)$ one needs to maintain a bipartition of $\bar{B}_{ST}$ into a pair of lists: $\bar{B}^0_{ST}$ (consisting of arcs with zero reduced weight) and $\bar{B}^1_{ST}$ (consisting of arcs with the reduced weight equal to 1). As only $O(n)$ dual updates are totally possible, the total time needed to deal with μ_4 is $O(n)$.

Therefore, each dual step totally takes $O(n \log n + n + m \log n) = O(m \log n)$ time, as claimed.

7 Conclusions

We have studied the minimum weight bibranching problem and devised an efficient combinatorial weight-scaling algorithm for it. Regardless of the fact that bibranchings do not seem to be reducible to network flows, our results indicate that this problem can benefit from the blocking augmentation strategy.

The major open question here is if the time bound of $O(m\sqrt{n} \log n \log(nW))$ can be further improved.

It seems likely that one may replace $m \log n$ term by $m + n \log n$. However, this would require using more involved data structures than just dynamic trees and meldable heaps. A good starting point for this might be [9] where an $O(m + n \log n)$ time algorithm for the optimal branching problem is given.

It is also tempting to improve the bound for the case when one part of the graph is much larger than the other. This change will require a more careful analysis of the number of primal and dual steps (as compared to what we did in Sect. 5).

Acknowledgements The author is thankful to Petr Mitrichev (Faculty of Mechanics and Mathematics, Moscow State University) and also to the anonymous referees for helpful comments and suggestions.

References

1. Babenko, M.: An efficient scaling algorithm for the minimum weight bibranching problem. Lect. Notes Comput. Sci. **5369**, 232–245 (2008)
2. Cormen, T., Stein, C., Rivest, R., Leiserson, C.: Introduction to Algorithms. McGraw-Hill Higher Education, New York (2001)

3. Dinic, E.: Algorithm for solution of a problem of maximum flow in networks with power estimation. Sov. Math. Dokl. **11**, 1277–1280 (1980)
4. Edmonds, J.: Optimum branchings. J. Res. Natl. Bur. Stand. **71B**, 233–240 (1967)
5. Frank, A.: How to make a digraph strongly connected. Combinatorica **1**(2), 145–153 (1981)
6. Fredman, M., Tarjan, R.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM **34**(3), 596–615 (1987)
7. Gabow, H.: A representation for crossing set families with applications to submodular flow problems. In: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 202–211 (1993)
8. Gabow, H.: A framework for cost-scaling algorithms for submodular flow problems. In: SFCS '93: Proceedings of the 1993 IEEE 34th Annual Foundations of Computer Science, pp. 449–458 (1993)
9. Gabow, H., Galil, Z., Spencer, T., Tarjan, R.: Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica **6**(2), 109–122 (1986)
10. Goldberg, A., Tarjan, R.: Solving minimum-cost flow problems by successive approximation. In: STOC '87: Proceedings of the Nineteenth Annual ACM Conference on Theory of Computing, pp. 7–18 (1987)
11. Gabow, H., Tarjan, R.: Faster scaling algorithms for network problems. SIAM J. Comput. **18**(5), 1013–1036 (1989)
12. Gabow, H., Tarjan, R.: Faster scaling algorithms for general graph matching problems. J. ACM **38**(4), 815–853 (1991)
13. Keijsper, J., Pendavingh, R.: An efficient algorithm for minimum-weight bibranching. J. Comb. Theory, Ser. B **73**(2), 130–145 (1998)
14. Schrijver, A.: Min-max relations for directed graphs. Ann. Discrete Math. **16**, 261–280 (1982)
15. Schrijver, A.: Combinatorial Optimization. Springer, Berlin (2003)
16. Sleator, D., Tarjan, R.: A data structure for dynamic trees. J. Comput. Syst. Sci. **26**(3), 362–391 (1983)
17. Tarjan, R.: Data Structures and Network Algorithms. Society for Industrial and Applied Mathematics, Philadelphia (1983)