Directed Shortest Paths via Approximate Cost Balancing*

James B. Orlin
MIT
jorlin@mit.edu†

László A. Végh
LSE
l.vegh@lse.ac.uk‡

Abstract

We present an $O(nm)$ algorithm for all-pairs shortest paths computations in a directed graph with n nodes, m arcs, and nonnegative integer arc costs. This matches the complexity bound attained by Thorup [26] for the all-pairs problems in undirected graphs. Our main insight is that shortest paths problems with approximately balanced directed cost functions can be solved similarly to the undirected case. Our algorithm starts with an $O(m\sqrt{n}\log n)$ preprocessing step that finds a 3-min-balanced reduced cost function. Using these reduced costs, every shortest path query can be solved in $O(m)$ time using an adaptation of Thorup's component hierarchy method. The balancing result is of independent interest, and gives the best currently known approximate balancing algorithm for the problem.

1 Introduction

Let $G = (N, A, c)$ be a directed graph with nonnegative arc costs, and $n = |N|$, $m = |A|$. In this paper, we consider the *single-source shortest paths (SSSP)* and the *all-pairs shortest paths (APSP)* problems. In the SSSP problem, the goal is to find the shortest paths from a given source node $s \in N$ to every other node; in the APSP problem, the goal is to determine the shortest path distances between every pair of nodes.

For the SSSP problem, there are two classic types of algorithms (see Ahuja et al. [1, Chapters 4-5]): *label correcting* and *label setting* algorithms. The fastest algorithms for the SSSP are label setting algorithms, including Dijkstra's classical 1959 algorithm [5]. Fredman and Tarjan [9] showed how to implement Dijkstra's algorithm in $O(m + n\log n)$ time via the development of a new data structured that they called Fibonacci heaps. Under the assumption that all of the arc lengths are integral, Thorup [27] improved the running time for the SSSP to $O(m+n\log\log n)$. Thorup's algorithm uses the word RAM model of computation, discussed in Section 2.

For the APSP problem, one can obtain $O(mn + n^2\log\log n)$ by running the SSSP algorithm of [27] n times. This has been the best previously known result for directed graphs. The main contribution of this paper is an $O(mn)$ algorithm for APSP in the word RAM model.

Our new result achieves the same complexity for APSP in directed graphs as in *undirected* graphs. A breakthrough result by Thorup [26] obtained a linear time SSSP algorithm in the word RAM model for undirected graphs, also implying $O(mn)$ for APSP. Our $O(mn)$ APSP algorithm is based on an $O(m\sqrt{n}\log n)$ preprocessing algorithm that enables SSSP queries in $O(m)$ time.

Thorup's [26] algorithm is a *label setting* algorithm that is similar to Dijkstra's algorithm. Label setting algorithms maintain upper bounds $D(i)$ on the true shortest path distances $d(i)$ from the origin node s to node i, and add nodes one-by-one to the set of permanent nodes S. At the time a node i is made permanent, $D(i) = d(i)$ holds. In Dijkstra's algorithm, the node that is made permanent is the one with the least value of $D(\,)$. That is, in the iteration when node i is made permanent, we have $D(i) \le D(j)$ for all other nodes $j \notin S$.

Let us define the *bottleneck costs* for nodes $i, j \in N$ as

(1.1)
$$b(i,j) = \min\left\{\max_{e\in P} c(e) : P \text{ is an } i\text{–}j \text{ path in } G\right\}.$$

Dinitz [6] showed that label setting algorithms are guaranteed to find the shortest path distances if the following is true whenever a node j is made permanent: for each node i that is not yet permanent, $D(j) \le D(i) + b(i,j)$. If an algorithm satisfies this weaker condition, then at termination it obtains distances satisfying $d(j) \le d(i) + b(i,j)$ for all i and j, which in turn implies the shortest path optimality conditions: $d(j) \le d(i) + c(i,j)$ for all $(i,j) \in A$.

*The full version can be found at https://arxiv.org/abs/2007.07975.

†Supported by the ONR Grant N00014–17–1–2194.

‡Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement ScaleOpt–757481).

Thorup's algorithm as well as the algorithm presented in this paper rely on this weaker guarantee of correctness. Both algorithms accomplish this by creating a component hierarchy; the variant we use is described in Section 2. Thorup developed the hierarchy framework for the SSSP on undirected networks. The hierarchy framework was extended to directed graphs in [12, 20, 21].

Our results also rely on the classical observation that shortest path computations are invariant under shifting the costs by a node potential. For a potential $\pi : N \to \mathbb{R}$, the *reduced cost* is defined as $c^\pi(u,v) = c(u,v) + \pi(u) - \pi(v)$. Computing shortest paths for c and any reduced cost c^π are equivalent: if P is a u–v path, then $c^\pi(P) = c(P) + \pi(u) - \pi(v)$.

We extend the use of reduced costs to the bottleneck costs.

$$b^\pi(i,j) = \min\left\{ \max_{e\in P} c^\pi(e) : P \text{ is an } i\text{–}j \text{ path in } G \right\}.$$

Our preprocessing step obtains a reduced cost function satisfying the following ξ-min-balancedness property for a constant $\xi > 1$.

DEFINITION 1.1. *A strongly connected directed graph $G = (N, A, c)$ with nonnegative arc costs $c \in \mathbb{R}_+^A$ and $\xi \geq 1$ is* ξ-min-balanced *if for any arc $e \in A$, there exists a directed cycle $C \subseteq A$ with $e \in C$, such that $c(f) \leq \xi c(e)$ for all $f \in C$.*

The importance of ξ-min-balancedness in the context of hierarchy-based algorithms arises from the near-symmetry of the bottleneck values $b(i,j)$: Lemma 2.1 below shows a graph is ξ-min-balanced if and only if $b(j,i) \leq \xi b(i,j)$ for all $i,j \in N$. Thorup's component hierarchy for undirected graphs implicitly relies on the fact that $b(i,j) = b(j,i)$ for all nodes i and j. For a ξ-balanced reduced cost function c^π, the values $b^\pi(i,j)$ and $b^\pi(j,i)$ are within a factor ξ. We can leverage this proximity to use component hierarchies essentially the same way as for undirected graphs in Thorup's original work [26], and achieve the same $O(m)$ complexity for an SSSP query, after an initial $O(m\sqrt{n}\log n)$ balancing algorithm.

This balancedness notion is closely related to the extensive literatue on matrix balancing and gives an improvement for approximate ℓ_∞-balancing. We give an overview of the related literature in Section 1.1.2.

1.1 Related work

1.1.1 Shortest path problem In the context of shortest path computations, the choice of the computational model is of high importance. The main choice is between the addition-comparison model with real costs, and variants of word RAM models with integer costs. In the addition-comparison model, the arithmetic operations of additions and comparisons each take $O(1)$ time, regardless of the quantities involved. The other mathematical operations are not permitted except in so much as they can be simulated using additions and comparisons.

There is an important difference between these computational models in terms of lower bounds: sorting in the addition-comparison model requires $\Omega(n \log n)$, whereas no superlinear lower bound is known for integer sorting. Since Dijkstra's algorithm makes nodes permanent in a non-decreasing order of the shortest path distance $d(i)$ from s, the $O(m+n\log n)$ Fibonacci-heap implementation [8] is an optimal implementation of Dijkstra's algorithm in the real addition-comparison model; this is in fact the best current running time of an SSSP algorithm in this model.

The best bound for APSP in the real addition-comparison model is $O(mn+n^2 \log\log n)$ by Pettie [21]. This matches the best previous running time bounds for the integer RAM model, where this was previously obtained in [12, 27].

Pettie's [21] algorithm is based on the hierarchy framework. The same paper gives a lower bound that, at first glance, seems to prove that an $O(m)$ running time for the directed SSSP is not achievable.

Let r be the ratio between the largest and the smallest nonzero arc cost. Pettie argued if a shortest path algorithm for the directed SSSP is based on the hierarchy framework, then the running time of the algorithm is $\Omega(m + \min\{n\log r, n\log n\})$, even if the hierarchy is already provided. His argument applied to analysis using word RAM model of computation as well as the real addition-comparison model. Moreover, his argument is valid even if the hierarchy framework is provided in advance. But Pettie's definition of the hierarchy framework for directed networks did not incorporate reduced costs. Therefore, his arguments do not contradict our development of an $O(m)$ time algorithm for the directed SSSP.

For undirected graphs, Pettie and Ramachandran [22] solve APSP in $O(mn\log\alpha(m,n))$ in the real addition-comparison model, where $\alpha(m,n)$ is the inverse Ackermann function. After an $O(m + \min\{n\log n, n\log\log r\})$ time preprocessing step, every SSSP problem can be solved in time $O(m\log\alpha(m,n))$.

For dense graphs, that is, graphs with $m = \Omega(n^2)$ edges, $O(nm) = O(n^3)$ can be achieved using the classical Floyd-Warshall algorithm [7, 28]. However, it is possible to achieve complexity $o(n^3)$. The first

such algorithm was given by Fredman [8], in time $O(n^3/\log^{1/3} n)$. This was followed by a long series of improvements with better logarithmic factors, see references in [29]. In 2014, Williams [29] achieved a breakthrough of $O(n^3/2^{\Omega(\log n)^{1/2}})$, by speeding up min-plus (tropical) matrix multiplication using tools from circuit complexity.

1.1.2 Approximate graph and matrix balancing

Our notion of ξ-min-balanced graphs is closely related to previous work on graph and matrix balancing. For $\xi = 1$, we simply say that G is min-balanced.

In Lemma 2.1, we will show that a graph G is min-balanced if and only if for each proper subset S of nodes, the following is true: the minimum cost over arcs entering S is at equal to the minimum cost over arcs leaving S.

Schneider and Schneider [23] defined max-balanced graphs where for every subset S, the maximum cost over arcs entering S equals the maximum cost over arcs leaving S. For each $e \in E$, let $c'(e) = c_{\max} - c(e)$. Then G is max-balanced with respect to c' if and only if G is min-balanced with respect to c. For exact min/max-balancing, the running time $O(mn+n^2 \log n)$ by Young, Tarjan, and Orlin [30] is still the best known complexity bound. Relaxing the exact condition, we can achieve the significantly better running time $O(\frac{m\sqrt{n}\log n}{\xi-1})$ for ξ-min-balancing for $1 < \xi$, $\xi = O(1)$.

Min-balancing extends the min-mean cycle problem: if C is a min-mean cycle, then any min-balanced residual cost function satisfies $c^\pi(e) \geq \mu$ for all $e \in E$ and $c^\pi(e) = \mu$ for $e \in C$ for some $\mu \in \mathbb{R}$. In fact, following [23], one can solve min-balancing as a sequence of min-mean cycle computations, see the discussion after Theorem 3.1. Karp's 1978 $O(mn)$ algorithm [13] is still the best known strongly polynomial algorithm for min-mean cycle problem. Weakly polynomial algorithms that run in $O(m\sqrt{n}\log(nC))$ time were given by Orlin and Ahuja [15] and by McCormick [14]. The latter provides a scaling algorithm based on the same subroutine of Goldberg [11] that plays a key role in our balancing algorithm. The algorithm in [14] easily extends to finding an ε-approximate min-mean cycle in $O(m\sqrt{n}\log(n/\varepsilon))$ time. That is, finding a reduced cost c^π, a cycle C, and a value μ such that $c^\pi(e) \geq \mu$ for all $e \in E$ and $c^\pi(e) \leq (1+\varepsilon)\mu$ for all $e \in C$.

A restricted case of the APSP is the problem of finding the shortest cycle in a network. Orlin and Sedeño-Noda [16] show how to solve the shortest cycle problem in $O(nm)$ time by solving a sequence of n (truncated) shortest path problems, each in $O(m)$ time. Their preprocessing algorithm was the solution of a minimum cycle mean problem in $O(nm)$ time. However—analogously to the approach in this paper—they could have relied instead on McCormicks algorithm to find a 2-approximation of the minimum cycle mean in $O(m\sqrt{n}\log n)$ time.

We say that a graph is *weakly max-balanced*, if for every node $v \in N$, the maximum cost over arcs entering v equals the maximum cost over arcs leaving v; that is, we require the property in the definition of max-balancing only for singleton sets $S = \{v\}$.

This notion corresponds to the well-studied *matrix balancing* problem: given a nonnegative matrix $M \in \mathbb{R}^{n\times n}$, and a parameter $p \geq 1$, find a positive diagonal matrix D such that in DMD^{-1}, the p-norm of the i-th column equals the p-norm of the i-th row. Given $G = (N, A, c)$, we let $M_{ij} = e^{c_{ij}}$ if $ij \in A$ and $M_{ij} = 0$ otherwise. Then, balancing M in ∞-norm amounts to finding a weakly max-balanced reduced cost c^π.

Matrix balancing was introduced by Osborne [17] as a natural matrix preconditioning for eigenvalue computations. He also proposed a natural iterative algorithm for ℓ_2-norm balancing. Parlett and Reinsch [19] extended this algorithm for other norms. Schulman and Sinclair [24] showed that a natural variant of the Osborne–Parlett–Reinsch (OPR) algorithm finds an ε-approximately balanced solution in ℓ_∞ norm in time $O(n^3 \log(n\rho/\varepsilon))$, where ρ is the initial imbalance. Ostrovsky, Rabani, and Yousefi [18] give polynomial bounds for variants of the OPR algorithm for fixed finite p values, in particular, $O(m + n^2\varepsilon^{-2}\log w)$ for a weighted randomized variant, where w is the ratio of the sum of the entries over the minimum nonzero entry, and m is the number of nonzero entries. Very recently, Altschuler and Parillo [3] showed an $\tilde{O}(m\varepsilon^{-2}\log w)$ bound for a simpler randomized variant of OPR. Cohen et al. [4] use second order optimization techniques to attain $\tilde{O}(m\log\kappa\log^2(nw/\varepsilon))$, where κ is the ratio between the maximum and minimum entries of the optimal rescaling matrix D; similar running times follow from [2]. The value κ may be exponentially large; [4] also shows a $\tilde{O}(m^{1.5}\log(nw/\varepsilon))$ bound via interior point methods using fast Laplacian solvers.[1]

Our graph balancing problem corresponds to ℓ_∞ matrix balancing. Except for [24], the above works are applicable for finite ℓ_p norms only. Compared to [24], our approximate balancing algorithm has lower polynomial terms, but our running time depends linearly on $1/\varepsilon$ instead of a logarithmic dependence.[2]

[1] In the quoted running times, $\tilde{O}(.)$ hides polylogarithmic factors. Various papers define ε-accuracy in different ways; here, we adapt the statements to ℓ_1-accuracy as in [3].

[2] We note that, in contrast to the previous work, we consider

1.2 Overview The rest of the paper is structured as follows. Section 2 introduces the notation and basic concepts, including the word RAM computational model, and the directed variant of component hierarchies used in this paper. Section 3 is dedicated to the approximate min-balancing algorithm. The algorithm is developed in several steps: a key ingredient is a subroutine by Goldberg [11] that easily gives rise to a weakly polynomial algorithm. In order to achieve a strongly polynomial bound, we need a further preprocessing step to achieve an initial 'rough balancing'. At the beginning of Section 3, we give a detailed overview of the overall algorithm and the various subsections.

In Section 4, we describe the shortest path algorithm for 3-min-balanced directed graphs. This is very similar to Thorup's original algorithm [26]. However, the setting is different, and we use a slightly different notion of the component hierarchy. We included a mostly complete yet concise description of the algorithm and the proof of correctness. Concluding remarks are given in the final Section 5.

Some technical details are deferred to the full version of the paper available at arxiv:2007:07975. From Section 3, these include the description of the new Union-Find-Increase data structure, and the precise adaptation of Goldberg's algorithm. From Section 4, some proofs and the details of the Split/FindMin data structure are omitted here.

2 Notation and preliminaries

For an integer k, we let $[k] = \{1, 2, \dots, k\}$. We let $\mathbb{Z}_+$ denote the nonnegative integers and let $\mathbb{Z}_{++}$ denote the positive integers; similarly for $\mathbb{Q}_+$ and $\mathbb{Q}_{++}$. We let $\log x = \log_2 x$ refer to base 2 logarithm unless stated otherwise.

Throughout, we let $G = (N, A, c)$ be a directed graph with nonnegative integer arc costs $c \in \mathbb{Z}_+^A$, and we let $n = \Theta(|N|)$ and $m = |A|$; we will always assume $n, m \geq 2$. All graphs considered will be simple and loopless. We write $n = \Theta(|N|)$ instead of $n = |N|$ since for the sake of the arithmetic model, it will be convenient to assume that $n = 2^t$ is an integer power of 2. Thus, we can define n as the smallest power of 2 greater or equal to $|N|$.

For a node $i \in N$, we let $A(i)$ denote the set of the outgoing arcs from i. For an arc set $F \subseteq A$, we let $N(F)$ denote the set of nodes incident to F. For a node set $X \subseteq N$, we let $A[X]$ denote the set of arcs in A with both endpoints inside X.

For a node set $S \subseteq N$, we let $\bar{S}$ denote the complement of S, i.e., $\bar{S} = N \setminus S$. We let $(S, \bar{S})$ denote the set of arcs directed from a node of S to a node of $\bar{S}$.

For a node set $Z \subseteq N$, we let G/Z denote the graph obtained by contracting Z. We let $G/Z = (N', A', c')$, where $N' = (N \setminus Z) \cup \{z\}$; here, z represents the contracted node set. We include every arc $(i, j) \in A$ in A' with the same cost if $i, j \notin Z$. Arcs with both endpoints in Z are deleted. If $i \in Z$ or $j \in Z$, the corresponding endpoint is replaced by z. In case parallel arcs are created, we only keep one with the smallest cost. For a partition $\mathcal{P} = (P_1, P_2, \dots, P_k)$ of N, the contraction $G/\mathcal{P}$ denotes the graph obtained after contracting each of the sets P_i, $i \in k$ in G; these contractions can be done in an arbitrary order.

We will assume that $G = (N, A, c)$ is *strongly connected*, that is, a directed path exists between any two nodes. If the input is not strongly connected, then we preprocess the graph as follows. We find the strongly connected components in $O(n+m)$ time using Tarjan's algorithm [25]. We select a value M greater than the sum of all arc costs, pick one node in each strongly connected component, add a directed cycle on these nodes, and set the cost of these arcs to M. This results in a strongly connected graph $G' = (N, A', c')$ with $|A'| = O(m + n)$. Computing shortest paths is equivalent in G' and in G; if the shortest path distance between nodes i and j in G' is at least M, then j is not reachable from i in G.

The word RAM model There is no universally accepted computational model for integer weights. We use the same as [12]; this is slightly more restrictive than the model in [26], which also allows unrestricted multiplications. Our model is the standard random access machine model, where every memory cell can store an integer of w bits, where we assume $w \geq \log n$. Unit-time operations include comparison, addition, subtraction, bit shifts by an arbitrary number of positions, bitwise boolean operations, and the most significant bit operation, that returns $\lfloor \log r \rfloor$ for an integer r. We do not allow multiplications and divisions in general. However, the bit shift operations enable multiplications by integer powers of 2 in $O(1)$ time. We can also simulate multiplications by an integer b using $O(\log b)$ addition and bit shift operations. Due to the assumption that n is a power of 2, multiplying by a monomial term such as bn^k can be done in $O(1)$ time if $b, k = O(1)$. Similarly, bit shift operations combined with bitwise boolean operations enable integer divisions by powers of 2.

For $r \in \mathbb{Z}_+$, we use the notation $\lfloor r \rfloor_2$ as the

min- rather than max-balancing. The exact min- and max-balancing problems can be transformed to each other by setting $c'(e) = c_{\max} - c(e)$; however, such a reduction does not preserve multiplicative approximation factors, and hence our result cannot be directly compared with [24]. Nevertheless, it seems that both algorithms can be adaptable to both the min and max settings. Such extensions are not included in this paper.

largest integer power of 2 smaller or equal than r; thus, $\lfloor r \rfloor_2 \leq r < 2\lfloor r \rfloor_2$. This can be implemented by the most significant bit operation.

The input cost vector c is given by integer costs. We assume that each input and output number fits into a single word; that is, both the input and the output can be described in size $O(m)$. Hence, our computational assumes that we can perform basic operations with the costs and distance labels in unit-time throughout the algorithm. In case this assumption does not hold, our algorithm could be modified to run in time linear in output size, as in [26].

Dijkstra's algorithm Dijkstra's algorithm [5] was the starting point of the fastest algorithms for the SSSP and the APSP. We now give a brief overview of the key steps. The algorithm maintains distance labels $D(i)$ for each node i that are upper bounds on $d(i)$, the shortest path distance from s. The algorithm adds nodes one-by-one to a *permanent node set* S with the property that $D(i) = d(i)$ for every $i \in S$. Further, for every $i \in N \setminus S$, $D(i)$ is the length of a shortest s–i path in the subgraph induced by the node set $S \cup \{i\}$.

These are initialized as $D(s) = 0$, $D(i) = \infty$ for $i \in N \setminus \{i\}$, and $S = \emptyset$. Every iteration adds a new node to S, selecting the node $i \in N \setminus S$ with the smallest label $D(i)$. Then, the algorithm considers every outgoing arc (i, j), and updates $D(j)$ to $\min\{D(j), D(i) + c(i, j)\}$. The crucial property of the analysis is that this selection rule is correct, that is, for $i \in \arg\min\{D(j) : j \in N \setminus S\}$, we must have $D(i) = d(i)$.

Bottleneck costs in balanced graphs Our shortest path algorithm will assume that the input graph is ξ-min-balanced (see Definition 1.1). An important consequence is that in such graphs, the bottleneck costs will also be approximately balanced, as shown next.

Recall the definition of the bottleneck cost $b(i, j)$ in (1.1). We extend the definition to non-empty disjoint subsets $S, T \subsetneq N$ as follows: $b(S, T) = \min\{b(i, j) : i \in S, j \in T\}$. Thus, $b(S, \bar{S}) = \min\{c(i, j) : i \in S, j \in \bar{S}\}$. By a *bottleneck* i–j path we mean an i–j path where the maximum arc cost is $b(i, j)$.

LEMMA 2.1. *The following are equivalent.*

(1) G is ξ-min-balanced.

(2) For all proper subsets $\emptyset \neq S \subsetneq N$, $b(\bar{S}, S) \leq \xi b(S, \bar{S})$.

(3) For all $i \in N$ and $j \in N$, $b(j, i) \leq \xi b(i, j)$.

Proof. **(1) ⇒ (2).** Suppose that G is ξ-min-balanced and suppose that $\emptyset \neq S \subsetneq N$. Choose $e = \text{argmin}\{c(e) : e \in (S, \bar{S})\}$. Thus, $c(e) = b(S, \bar{S})$. Let C be the bottleneck cycle containing e. Because C contains an arc f of $(\bar{S}, S)$, the following is true: $b(\bar{S}, S) \leq c(f) \leq \xi c(e) = \xi b(S, \bar{S})$.

(2) ⇒ (3). Suppose that (2) is true. For given nodes i and j, let $S = \{k \in N : b(j, k) \leq \xi b(i, j)\}$. Clearly, $j \in S$. We claim that $i \in S$, and thus $b(j, i) \leq \xi b(i, j)$. We will show that $i \in S$ via a contradiction. Suppose that $i \in \bar{S}$. Let $e = \text{argmin}\{c(e) : e \in (S, \bar{S})\}$, and suppose that $e = (h, \ell)$. Then $b(j, h) \leq \xi b(i, j)$ because $h \in S$. And $c(h, \ell) = b(S, \bar{S}) \leq \xi b(\bar{S}, S) \leq \xi b(i, j)$ by (2) and the fact that the bottleneck path from i to j includes an arc of $(\bar{S}, S)$. Then $b(j, \ell) \leq \max\{b(j, h), c(h, \ell)\} \leq \xi b(i, j)$. But this implies that $\ell \in S$, which is a contradiction.

(3) ⇒ (1). Suppose that (3) is true. Let $e = (j, i)$ be any arc of A. Note that $b(j, i) \leq c(e)$. Let P be a path from i to j with arcs of length at most $b(i, j)$, and let $C = P \cup \{e\}$. Then C is a cycle, and $\max\{c(f) : f \in C\} \leq \max\{b(i, j), c(e)\} \leq \max\{\xi b(j, i), c(e)\} \leq \xi c(e)$. Thus, G is ξ-min-balanced. ∎

The component hierarchy We now introduce the concept of the component hierarchy. This is a variant of Thorup's [26] component hierarchy, adapted for approximately min-balanced directed graphs. The papers [12, 20, 21] also use component hierarchies for directed graphs. However, our notion exploits the ξ-min-balanced property, and will be more similar to the undirected concept [26] in that it does not impose orderings of the children of the vertices.

The definition uses the standard terminology for rooted trees. Consider a tree (V', E') rooted at $r \in V'$.

- For $v \in V' \setminus \{r\}$, the *parent* $\text{p}(v)$ of v is the first vertex after v on the unique path in the tree from v to r. All nodes in the path are called the *ancestors* of v.
- For $v \in V'$, $\text{children}(v) \subseteq V'$ is the set of nodes u such that $\text{p}(u) = v$.
- For $u, v \in V'$, $\text{lca}(u, v)$ is the least common ancestor of u and v, i.e. the unique vertex on the u–v path in E' that is an ancestor of both u and v.

DEFINITION 2.1. *Given a strongly connected directed graph $G = (N, A, c)$ and value $\lambda > 1$, $(V \cup N, E, r, a)$ is called the* component hierarchy of G with parameter λ *if*

- *$(V \cup N, E)$ is a tree with root $r \in V$, and N is the set of leaves. The vector $a : V \to \mathbb{Q}_+$ is an integer power of λ: $a(v) = \lambda^{t(v)}$ for some $t(v) \in \mathbb{Z}_+$.*
- *For every $v \in V$, the set $\text{desc}(v) \subseteq N$ is the set of leaves of the subtree rooted at v. The subgraph induced by $\text{desc}(v)$ is strongly connected.*

- *For any $i, j \in N$ with $\mathrm{lca}(i,j) = v$, we have $a(v) \le b(i,j) \le (2\lambda-1)a(v)$; moreover, there exists a bottleneck i–j path entirely contained inside the subgraph induced by the node set* $\mathrm{desc}(v)$.

3 An algorithm for approximate min-balancing

This section is dedicated to the proof of the following theorem. The algorithm asserted in the theorem is presented in Algorithm 3.

THEOREM 3.1. *Assume we are given a strongly connected directed graph $G = (N, A, c)$ with nonnegative integer arc costs $c \in \mathbb{Z}_+^A$, and parameters $\xi > 1$, $\xi = O(1)$, and $\lambda = (\xi+1)/2$. There exists an $O\left(\frac{n^{1/2} m \log n}{\lambda - 1}\right)$ time algorithm in the word RAM model that finds a potential $\pi \in \mathbb{Q}^N$ such that c^π is ξ-min-balanced. For $\xi = 3$ and $\lambda = 2$, in the same running time we can also obtain a component hierarchy of (N, A, c^π) with parameter 2.*

We note that a component hierarchy can also be obtained for other parameter values, using a more permissive arithmetic model, including multiplication and division. The definition of the component hierarchy requires the $a(v)$ values to be integer powers of the (possibly fractional number) λ. For $\lambda = 2$, we can essentially work with integers and only use divisions amounting to bit-shifts, but for fractional λ values, more careful rational arithmetics would be needed. We do not present such a variant as the component hierarchy will be only required for $\lambda = 2$ in Section 4.

To give an overview of the algorithm, it is instructive to start from the problem of exact min-balancing, that is, $\xi = 1$, even though our algorithm is not applicable to this case. For $\xi = 1$, the exact max-balancing algorithms [23, 30] can be used (by negating the costs). The simple and natural algorithm (see [23]) is based on the iterative application of min-mean cycle finding. First, we find all arcs that are in a min-mean cycle in the graph; let $\mu \ge 0$ denote the minimum cycle mean value, and F the set of all arcs in such cycles. Every arc $e \in F$ must have $c^\pi(e) = \mu$ if c^π is a min-balanced reduced cost function. To see this, note that for every cycle C and every potential π, $c^\pi(C) = c(C) \ge |C|\mu$. If $c^\pi(e) < \mu$ for an arc in a min-balanced reduced cost function, then it would need to be contained in a cycle C with $c^\pi(C) < \mu|C|$, a contradiction. The equality $c^\pi(C) = |C|\mu$ for minimum mean cycles then shows that $c^\pi(e) = \mu$ must hold for every edge in C.

In fact, the minimum cycle mean finding algorithm produces a potential π such that $c^\pi(e) \ge \mu$ for all $e \in E$, and $c^\pi(e) = \mu$ for all $e \in F$. We can then contract all connected components of F, and recurse on the contracted graph, by repeatedly modifying the potential π and contracting the components of minimum-mean cycles.

The current best running time for min-balancing is $O(mn + n^2 \log n)$ [30]. The current best running time for a single minimum-mean cycle computation is $O(mn)$ [13]. Both of these running times are substantially higher than the overall running time given in Theorem 3.1.

We can thus only afford to approximately compute min-mean cycles. This can be achieved faster, using a subroutine in Goldberg's paper [11], originally developed for a weakly polynomial algorithm for negative cycle detection. Our variant of this subroutine, SMALL-CYCLES, is introduced in Section 3.1. There are some minor technical differences from [11]; the detailed description of the subroutine and the proof of correctness are given in the full version of the paper.

The input to the subroutine SMALL-CYCLES is a strongly connected directed graph with minimum arc cost L. In time $O(m\sqrt{n})$, we can identify strongly connected components of arcs with reduced cost in the range of $[L, \xi L]$, while also finding a new potential such that the reduced arc of any cost between different connected components is at least λL (recall $\lambda = (\xi + 1)/2$).

If the input graph has positive arc costs, the iterative application of this subroutine yields a simple weakly polynomial algorithm with running time $O\left(\frac{m\sqrt{n}\log(nC_{\max}/C_{\min})}{\lambda-1}\right)$, as described in Section 3.2.

In order to turn this into a strongly polynomial algorithm, we first devise a preprocessing algorithm to find a $7n^2$-min-balanced reduced cost in Section 3.3. The main part of this algorithm is determining the *balance values* $\beta(e)$ for all arcs $e \in E$; this is defined as the smallest value b such that G contains a cycle C with $e \in C$ and $c(f) \le b$ for all $f \in C$. These balance values can be efficiently found using a a divide-and-conquer framework. The potential π achieving a 'rough' balancing can be defined using the balance values. As a simple illustration of this 'rough' balancing, assume the entire graph is a cycle $(v_1, v_2, \ldots, v_n, v_1)$ with $c(v_n, v_1) = b \ge n$ and $c(v_i, v_{i+1}) = 1$ for $i \in [n-1]$. Thus, $\beta(e) = b$ for every arc. Setting $\pi(i) = ib/n$ for all $i \in [n]$, we get $c^\pi(v_i, v_{i+1}) = 1 + b/n$ for $i \in [n-1]$ and $c^\pi(v_n, v_1) = b/n$, making the graph 2-balanced.

The strongly polynomial algorithm asserted in Theorem 3.1 is given in Section 3.4. Here, the input is assumed to be a graph with a $7n^2$-min-balanced cost-function. How can we benefit from this 'rough' balance of the input? The weakly polynomial algorithm consists of $O\left(\frac{\log(nC_{\max}/C_{\min})}{\lambda-1}\right)$ calls of SMALL-CYCLES. Each call of SMALL-CYCLES has as input all of the arcs of

A that have yet to be contracted. For this reason, the running time is $O(m\sqrt{n})$ per call. However, when running Small-Cycles in which the lower bound is L, it is possible to restrict attention to arcs e with $c(e) \leq 2nL$. We refer to such arcs as 'active'. If the input is assumed to be a $7n^2$-min-balanced cost-function, then each arc is active for $O\left(\frac{\log n}{\lambda-1}\right)$ calls of Small-Cycles prior to being contracted. Thus each arc contributes $O\left(\frac{\sqrt{n}\log n}{\lambda-1}\right)$ to the total running time.

In the weakly polynomial algorithm, the parameter L giving a lower bound on the minimum reduced cost of non-contracted arcs increases by a factor λ in each iteration. To avoid the dependence on $C_{\max}/C_{\min}$, in the strongly polynomial algorithm this value may sometimes 'jump' by large amounts in cases where there are no currently active arcs.

An important technical detail is the maintenance of the reduced costs. In every iteration, we only directly maintain $c^\pi(e)$ for the current active arcs. Querying the reduced cost of a newly activated arc is nontrivial, since one or both of its endpoints may have been part of one or more contracted cycles, each of which leads to a new node in the contracted graph. To compute the potential of an original node i, we need to add to the potential of node i the potentials of every contracted node j that contains node i. This requires a new extension of the Union-Find datastructure, called Union-Find-Increase by incorporating a new 'increase' operation. This is described in the full version.

Preprocessing and contractions We will use contractions several times during our algorithms. Whenever a set S is contracted, we let s be the contracted node, and we set the potential $\pi_s = 0$. For each arc with one endpoint in S, we keep the same reduced cost as immediately before the contraction.

On multiple occasions we will need the subroutine Strongly-connected(N, A) that implements Tarjan's algorithm [25] to find the strongly connected components of the directed graph (N, A) in time $O(|N| + |A|)$. The output includes the strongly connected components (N_1, A_1), $(N_2, A_2), \ldots, (N_k, A_k)$ in the topological order, namely, for an arc $(u, v) \in A$ such that $u \in N_i$, $v \in N_j$, it must hold that $i \leq j$.

In Theorem 3.1, the input is a nonnegative integer cost function. For our algorithm, it is more convenient to assume a strictly positive cost function. We now show how the nonnegative case can be reduced to the strictly positive case by a simple $O(m)$ time preprocessing. Let $C_{\max}$ denote the largest value of the nonnegative integer cost function.

We first call Strongly-connected(N, A_0) on the subgraph of 0-cost arcs A_0. We contract all strongly connected components, and keep the notation $G = (N, A)$ for the contracted graph, where the output of the subroutine gives a topological ordering $N = \{v_1, v_2, \ldots, v_n\}$ such that for every 0-cost arc (v_i, v_j), we must have $i < j$. Let us set $\pi(v_i) = -i/n$. Then, it is easy to see that $c^\pi(e) \geq 1/n$ for every $e \in A$.

We then replace the cost function c by nc^π; this can be done in $O(m)$ time, since a multiplication by n can be implemented by a bit-shift by recalling the assumption that n is a power of 2. This finishes the description of the preprocessing step, after which we obtain an integer cost function with $1 \leq c(e) \leq nC_{\max}+n$ for every $e \in A$.

3.1 Approximate minimum-mean cycles: Goldberg's algorithm

The next theorem summarizes the properties of the subroutine Refine in Goldberg's paper [11].

Theorem 3.2. ([11]) *Let $G = (N, A, c)$ be a directed graph with an integer cost function $c \in \mathbb{Z}^A$ such that $c(e) \geq -1$ for all $e \in A$, and $|N| = n$, $|A| = m$. Then, in $O(m\sqrt{n})$ time, one can either find a negative cost cycle $C \subseteq A$, or an integer valued potential vector π such that $c^\pi(e) \geq 0$ for all $e \in A$.*

Goldberg uses this subroutine in a scaling framework to either find a negative cycle or a potential with nonnegative reduced cost. This subroutine runs in time $O(\sqrt{n}m \log U)$, where U is the largest absolute value of the most negative arc cost in the integer cost function. We slightly strengthen Goldberg's result to implement the following subroutine.

Subroutine Small-cycles
Input: A directed graph $G = (N, A, c)$ with a cost function $c \in \mathbb{Q}^A_+$, and $L, D \in \mathbb{Q}_+$ such that $c(e) \geq L$ for all $e \in A$.
Output: A partition $\mathcal{P} = (P_1, P_2, \ldots, P_k)$ of the node set N and a potential vector $\pi \in \mathbb{Q}^N$ such that

(i) For every $i \in [k]$, $c^\pi(e) \geq L$ for every $e \in A[P_i]$, and P_i is strongly connected in the subgraph of arcs $\{e \in A[P_i] : L \leq c^\pi(e) \leq L + 2D\}$.

(ii) $c^\pi(e) \geq L + D$ for all $e \in A \setminus \left(\cup_{i\in[k]} A[P_i]\right)$,

(iii) $-|N|D \leq \pi(v) \leq 0$, and $\pi(v)$ is an integer multiple of D for all $v \in N$.

Lemma 3.1. Small-cycles (L, D, N, A, c) *can be implemented in $O(|A|\sqrt{|N|})$ time.*

The proof is given in the full version. We use Goldberg's algorithm for the cost function $\bar{c}(e) = \left\lfloor \frac{c(e)-L}{D} \right\rfloor - 1$. The main difference is that in case a negative cycle is found, the original algorithm terminates, whereas we proceed after contracting the node set of the cycle; these cycles will be used to construct the output partition classes P_i.

3.2 A simple weakly polynomial variant We now present Algorithm 1, a weakly polynomial $O\left(\frac{m\sqrt{n}\log(nC_{\max}/C_{\min})}{\lambda-1}\right)$ time algorithm. Here, $C_{\max}$ and $C_{\min}$ denote the largest and smallest values of the strictly positive cost function, and $\lambda = (\xi+1)/2 > 1$ as in Theorem 3.1.

We initialize $L_1 = C_{\min}$. Every iteration calls Small-Cycles for the current value of L_t and $D_t = (\lambda-1)L_t$. In Step 6, we contract each subset (some or all of which may be singletons) in the partition $\mathcal{P}_t$ returned by the subroutine, and iterate with the returned reduced cost, setting the new value $L_{t+1} = L_t + D_t = \lambda L_t$.

Algorithm 1 Simple-min-balance

Input: A strongly connected directed graph $G = (N, A, c)$ with positive arc costs $c \in \mathbb{Q}^A_{++}$, and $\xi > 1$, $\lambda = (\xi+1)/2$.

Output: A potential $\pi \in \mathbb{Q}^N$ such that c^π is ξ-min-balanced.

1: $(\hat{N}_1, \hat{A}_1, \hat{c}_1) \leftarrow (N, A, c)$; $t \leftarrow 1$;
2: $L_1 \leftarrow \min_{e\in A} c(e)$;
3: **while** $|\hat{N}_t| > 1$ **do**
4: $\quad D_t \leftarrow (\lambda-1)L_t$;
5: $\quad (\mathcal{P}_t, p_t) \leftarrow$ Small-Cycles$(L_t, D_t, \hat{N}_t, \hat{A}_t, \hat{c}_t)$;
6: $\quad (\hat{N}_{t+1}, \hat{A}_{t+1}, \hat{c}_{t+1}) \leftarrow (\hat{N}_t, \hat{A}_t, \hat{c}_t^{p_t})/\mathcal{P}_t$;
7: $\quad L_{t+1} \leftarrow \lambda L_t$; $t \leftarrow t+1$;
8: Uncontract $(\hat{N}_t, \hat{A}_t, \hat{c}_t)$, and compute the overall potential $\pi : N \to \mathbb{Q}$;
9: **return** π.

We let $(\hat{N}_t, \hat{A}_t)$ denote the contracted graph at iteration t. The algorithm terminates when $\hat{N}_t$ has a single node only, at iteration $t = T$.

Uncontraction In line 8, we uncontract all sets in the reverse order of contractions. We start by setting $\pi = p_T$. Assume a set S was contracted to a node s in iteration t, and we have uncontracted all sets from iterations $t+1, \dots, T$. When uncontracting S, for every $v \in S$ we set $\pi(v) = p_t(v) + \pi(s)$, i.e., the potential right before contraction, plus the potential of s accumulated during the uncontraction steps. This takes time $O(n')$ where n' is the total size of all sets contracted during the algorithm; it is easy to bound $n' \le 2n$. Thus, the total time for uncontraction is $O(n)$.

Lemma 3.2. *Given a strongly connected digraph with positive arc costs and $\xi > 1$, $\xi = O(1)$, and $\lambda = (\xi+1)/2$, Algorithm 1 finds a ξ-min-balanced cost function in $O\left(\frac{m\sqrt{n}\log(nC_{\max}/C_{\min})}{\lambda-1}\right)$ iterations, where $C_{\min}$ and $C_{\max}$ are the smallest and largest arc costs in the input.*

Proof. At initialization, $L_1 = C_{\min}$, and L_t increases by a factor λ in every iteration. At every iteration, we can extend the cost function $\hat{c}_t$ to the original arc set A: for an arc e contracted in an earlier iteration $\tau < t$, we let $\hat{c}_t(e) = \hat{c}_\tau(e)$ represent the value right before the contraction. It is easy to see that this extension of $\hat{c}_t$ to A gives a valid reduced cost of c.

Throughout, we have that $L_t \le \hat{c}_t(e)$ for all $e \in \hat{A}_t$, and $\hat{c}_t(e) \ge 0$ for all contracted arcs. Thus, for any cycle $C \subseteq A$ that contains some non-contracted arcs in $\hat{A}_t$, we have $2L_t \le \hat{c}_t(C) = c(C) \le nC_{\max}$. Consequently, we have $L_t \le nC_{\max}/2$ throughout, implying the bound $O(\log_\lambda(nC_{\max}/C_{\min}))$. For $\lambda = O(1)$, this gives $O\left(\frac{\log(nC_{\max}/C_{\min})}{\lambda-1}\right)$.

As explained above, the final uncontraction and computing π can be implemented in $O(n)$ time.

To show that the final c^π is ξ-min-balanced, consider an arc $e \in A$, and assume it was contracted in iteration t, that is, $e \in A[P_j]$ for a component P_j of the partition $\mathcal{P}_t$. In particular, $c^\pi(e) = c^{p_t}(e) \ge L_t$. The set P_j is strongly connected in the subgraph of arcs of reduced cost $\le L_t + 2D_t = \xi L_t$. Thus, at iteration t, P_j contains a cycle C with $e \in C$ such that $\hat{c}_t(f) \le \xi L_t$ for all $f \in C$. This cycle may contain nodes that were contracted during previous iterations. Every component previously contracted contains a strongly connected subgraph of arcs with costs $< \lambda L_t$, noting that the arc costs do not change anymore after contraction. Thus, when uncontracting a node we can extend C to a cycle of arc costs $< \xi L_t$. Hence, we can obtain a cycle C' in the original graph G with $e \in C'$ and $c^\pi(f) \le \xi L_t \le \xi c^\pi(e)$ for all $f \in C'$. ∎

The algorithm uses operations permitted in the word RAM model for $\xi = 3$ and $\lambda = 2$. For smaller values of λ, the multiplication by λ in Step 7 may not be a permissible operation of the word RAM model. The strongly polynomial variant Algorithm 3 in in Section 3.4 includes modifications to conform the word RAM model; these are omitted here for simplicity.

3.3 A quick algorithm for rough balancing In this section, we present a subroutine Rough-balance(N, A, c) that finds a potential $\pi \in \mathbb{Q}^N$ such that c^π is $7n^2$-min-balanced. As mentioned previously, this will be an important preprocessing for the strongly polynomial algorithm in Section 3.4. The running time

can be stated as follows.

LEMMA 3.3. *Let $G = (N, A, c)$ be a strongly connected directed graph with $c \in \mathbb{Q}^N_{++}$. Then, in time $O(m \log^2 n)$, we can find a potential $\pi \in \mathbb{Q}^N$ such that c^π is $7n^2$-min-balanced, where $n = |N|$ and $m = |A|$.*

Given $G = (N, A, c)$ with $c \in \mathbb{R}^A_+$, and $r > 0$, we let $G[\leq r]$ denote the subgraph of G formed by the arcs $e \in A$ with $c(e) \leq r$. Similarly, we let $G[< r]$ denote the subgraph with arcs $c(e) < r$. For every $e \in A$, we define $\beta(e) \in \mathbb{Q}_{++}$ as the smallest value r such that $G[\leq r]$ contains a directed cycle C with $e \in C$. We call $\beta(e)$ the *balance value* of e. Clearly, G is ξ-min-balanced if and only if $\beta(e) \leq \xi c(e)$ for each $e \in A$.

The algorithm proceeds in two stages. First, we present FIND-BALANCE(N, A, c) (Section 3.3.1) that determines the balance value $\beta(e)$ for each arc in $e \in A$. The main algorithm ROUGH-BALANCE(N, A, c) (Section 3.3.2) relies on these values to find a potential $\pi \in \mathbb{Q}^N$ such that c^π is $7n^2$-balanced.

3.3.1 Determining the balance values

Algorithm 2 presents the subroutine FIND-BALANCE(N, A, c). We first sort the arcs in non-decreasing order of cost; we let e_i denote the i-th arc in this order. First, in lines 2-7, we identify the smallest value $r = c(e_h)$ such that the strongly connected components of $G[\leq r]$ contain at least half of the arcs. This can be found via a binary search on the $c(e_i)$ values, and repeatedly calling the subroutine STRONGLY-CONNECTED. We let $(U_1, F_1), (U_2, F_2), \ldots, (U_k, F_k)$ denote the strongly connected components of $G[< r]$ and $(N_1, A_1), (N_2, A_2), \ldots, (N_s, A_s)$ denote the strongly connected components of $G[\leq r]$ in lines 9 and 10. These are two partitions of N, i.e., $N = \cup_{i=1}^k U_i = \cup_{j=1}^s N_j$. Also note that $F_i \subseteq A[U_i]$ and $A_j \subseteq A[N_j]$, but these containments may be strict. (We note that there is no need for additional calls to STRONGLY-CONNECTED in lines 9 and 10. These partitions have already been found during the binary search, with the possible exceptions of $h = 1$, in which case $k = |N|$ and the U_i's are all singletons, and $h = m$, in which case $s = 1$ and $(N_1, A_1) = (N, A)$.)

The partition $(U_1, U_2, \ldots, U_k)$ is a refinement of the partition $(N_1, N_2, \ldots, N_s)$, that is, each set U_i is a subset of some set N_j. For all arcs $e \in A[N_j] \setminus \left(\bigcup_{i:U_i \subseteq N_j} F_i\right)$, we set $\beta(e) = \max\{r, c(e)\}$ in line 11.

We make $k + 1$ recursive calls to FIND-BALANCE. For each $i \in [k]$, we call the subroutine for the subgraph (U_i, F_i), and obtain balance values $\beta_i : F_i \to \mathbb{Q}$. Further, we call the subroutine for the k-node graph $\hat{G}$ obtained by contracting all sets N_j, returning balance values $\hat{\beta} : \hat{A} \to \mathbb{Q}$. If $e \in F_i$ for some $i \in [k]$, then we set $\beta(e) = \beta_i(e)$. If $e \in A \setminus \left(\cup_{j=1}^s A[N_j]\right)$, then we get $\beta(e)$ as the value obtained from the recursive call to $\hat{G}$ for the contracted image of e. We show that these correctly determines the balance values.

Algorithm 2 FIND-BALANCE

Input: A strongly connected directed graph $G = (N, A, c)$ with $c \in \mathbb{Q}^A_{++}$.
Output: A function $\beta : A \to \mathbb{Q}$ giving the balance value $\beta(e)$ of each arc $e \in A$.

1: Sort the arc set A in increasing order of the cost such that $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$;
2: $\ell \leftarrow 0$, $h \leftarrow m$;
3: **while** $h - \ell > 1$ **do**
4: $\quad t \leftarrow \left\lceil \frac{\ell + h}{2} \right\rceil$;
5: $\quad \{(N_1, A_1), (N_2, A_2), \ldots, (N_k, A_k)\}$ $\leftarrow$STRONGLY-CONNECTED$(G[\leq c(e_t)])$;
6: $\quad$ **if** $\sum_{i=1}^k |A_i| \geq |A|/2$ **then** $h \leftarrow t$;
7: $\quad$ **else** $\ell \leftarrow t$;
8: $r \leftarrow c(e_h)$;
9: $\{(U_1, F_1), (U_2, F_2), \ldots, (U_k, F_k)\}$ $\leftarrow$ STRONGLY-CONNECTED$(G[< r])$;
10: $\{(N_1, A_1), (N_2, A_2), \ldots, (N_s, A_s)\}$ $\leftarrow$ STRONGLY-CONNECTED$(G[\leq r])$;
11: **for** $e \in \left(\bigcup_{j=1}^s A[N_i]\right) \setminus \left(\bigcup_{i=1}^k F_i\right)$ **do** $\beta(e) \leftarrow \max\{r, c(e)\}$;
12: **for** $i = 1, \ldots, k$ **do**
13: $\quad$ **if** $|U_i| > 1$ **then**
14: $\quad\quad \beta_i \leftarrow$FIND-BALANCE$(U_i, F_i, c|_{F_i})$;
15: $\quad\quad$ **for** $e \in F_i$ **do** $\beta(e) \leftarrow \beta_i(e)$;
16: Obtain $\hat{G} = (\hat{N}, \hat{A}, \hat{c})$ by contracting N_j to $\hat{v}_j$ for all $j \in [s]$.
17: $\hat{b} \leftarrow$ FIND-BALANCE$(\hat{G})$;
18: **for** $e \in A \setminus \left(\bigcup_{j=1}^s A[N_j]\right)$ **do** $\beta(e) \leftarrow \hat{\beta}(\hat{e})$, where $\hat{e}$ is the contracted image of e ;
19: **return** β.

LEMMA 3.4. *Algorithm 2 correctly computes the balance values in G in time $O(m \log^2 n)$.*

Proof. First, we show correctness by induction on $|N|$. Let us start with arcs $e \in A[N_j] \setminus \left(\bigcup_{i:U_i \subseteq N_j} F_i\right)$ for some $j \in [s]$, where we set $\beta(e) = \max\{r, c(e)\}$. We show that this is indeed the correct choice. First, e is not contained in any cycle of $G[< r]$, and thus $\beta(e) \geq r$. Second, $A_j \cup \{e\}$ contains a cycle C with $e \in C$, and $c(f) \leq r$ for any $f \neq e$, since (N_j, A_j) is strongly connected in $G[\leq r]$. Hence, $\beta(e) \leq \max\{r, c(e)\}$.

If $e \in F_i$ for some $i \in [k]$, we set $\beta(e) = \beta_i(e)$ from the recursive call to (U_i, F_i). This is correct since

$\beta_i(e) < r$; hence, arcs outside F_i cannot contribute to a cycle with smaller maximum cost.

Finally, let $e \in A \setminus \left(\bigcup_{j=1}^{s} A[N_j]\right)$, and let $\hat{e} \in \hat{A}$ be the image of e in the contracted graph $\hat{G}$. By the inductive hypothesis, in the recursive call to $\hat{G}$, we find a value $\hat{\beta}(\hat{e})$ such that there exists a cycle $\hat{C}$ with $\hat{e} \in \hat{C} \subseteq \hat{A}$, $\hat{c}(\hat{f}) \leq \hat{\beta}(\hat{e})$ for all $\hat{f} \in \hat{C}$. Furthermore, $\hat{\beta}(\hat{e})$ is the smallest value with this property. This cycle can be mapped back to a cycle $C \subseteq A$ by connecting the endpoints of the pre-images of the arcs in $\hat{C}$ via paths in F_j. Then, $\beta(e) = \hat{\beta}(\hat{e})$ follows from the following facts: each (N_i, A_i) is strongly connected, all arc costs in A_i are $\leq r$, and $\hat{\beta}(\hat{e}) > r$, since every cycle in $\hat{G}$ must contain an arc of cost greater than r.

We now turn to the recursive estimation of the running time bound. Recall that $\log x$ refers to base 2 logarithm. The binary search makes at most $\log m$ calls to Strongly-Connected for graphs with n nodes and at most m arcs. Since the input graph is strongly connected, we have $n \leq m$. Consequently, the running time to these calls is $O((n+m)\log m) = O(m \log m)$. This dominates the running time of all operations except the recursive calls. Thus, there exists a constant $\alpha > 0$ such that the running time of all operations except the recursive calls can be bounded as $\alpha m \log m$.

By induction on m, we show that the total running time is $\alpha m \log^2 m$. Let $m_i = |F_i|$ for $i \in [k]$. By induction, if $|U_i| > 1$, then the running time of the recursive call to $G[U_i]$ is $\alpha m_i \log^2 m_i$. By the choice of r, $\sum_{i=1}^{k} m_i < m/2$. Therefore the total running time of these calls is $\leq \alpha(m/2)\log^2(m/2) = \alpha(m/2)(-1 + \log m)^2$.

Also by the choice of r, the graph $\hat{G}$ has $\leq m/2$ arcs; thus, the same bound applies for the running time of the recursive call to $\hat{G}$. Thus the total running time is at most

$$\alpha m(-1+\log m)^2 + \alpha m \log m < \alpha m \log^2 m\,.$$

Finally, the running time bound $O(m \log^2 n)$ follows since we assumed the graph is simple, and thus $\log m = O(\log n)$. □

3.3.2 Constructing the potential We now describe the algorithm Rough-balance(N, A, c). We first obtain the balance values $\beta(e)$ by running Find-Balance(N, A, c).

To motivate the algorithm, we first illustrate how how one might roughly balance the arcs $e \in A[N_j] \setminus \left(\bigcup_{i:U_i \subseteq N_j} F_i\right)$ in the Find-Balance algorithm. Recall that (N_j, A_j) are the strongly connected component of $G[\leq r]$, and (U_i, F_i), $i \in [k']$ are the strongly connected components of $G[< r]$ with $U_i \subseteq N_j$. Assume these components are in the topological order, that is, for every $(u, v) \in A$ with $u \in U_i$ and $v \in U_{i'}$ and $c(e) < r$ we have $i \leq i'$.

Every arc $e \in A[N_j] \setminus \left(\bigcup_{i:U_i \subseteq N_j} F_i\right)$ has $\beta(e) = \max\{r, c(e)\}$. If we set $\pi(v)$ by ri/n for every $v \in U_i$ and $i \in [k']$, then for every arc $e \in A[N_j]$, $c(e) < r$, the reduced cost will become $c^\pi(e) \geq r/n$. The reduced cost $c^\pi(e)$ may also decrease for arcs $e = (u, v)$ where $u \in U_i$ and $v \in U_{i'}$ for $i > i'$; however, $c(e) \geq r$ for all such arcs, and the amount of decrease is bounded by $(n-1)r/n$. Hence, we will have $c^\pi(e) \geq r/n$ also for these arcs.

There are two problematic issues with extending this idea to include the remaining arcs. One is correctness: potential adjustments made for different values of r may interfere: in the above example, such adjustments would be done inside each set U_i for lower r values, and could even result in negative reduced costs. The other issue is running time: we cannot afford to make such a potential change for each different $\beta(e)$ values, as there may be $\Omega(n)$ such values. To address both these problems, we devise an iterative scheme that makes adjustments for a subset of $\beta(e)$ values at each iteration.

In iteration $t = 1, 2, \ldots, T$, we maintain a contraction $(\hat{N}_t, \hat{A}_t)$ of the input graph (N, A), as well as an 'active' arc set $F_t \subseteq \hat{A}_t$. We handle contractions and maintain a reduced cost c^π as in Section 3.2. That is, the final reduced cost of an arc e will be equal to its reduced cost immediately before its endpoints got contracted into the same node. The algorithm terminates at iteration $T = t$ when $\hat{N}_T$ becomes a singleton. Then we uncontract and obtain the overall potential and reduced cost in the original graph in time $O(n)$. Throughout, we identify an arc $e \in A$ with its images in the A_t sets; that is, $A = \hat{A}_1 \supseteq \ldots \supseteq \hat{A}_T$.

For each iteration t, we define a threshold value r_t, defined as follows.

(3.2)

$$r_1 := \min\{\beta(e) : e \in A\}\,,$$

$$r_{t+1} := \begin{cases} 2nr_t\,, & \text{if } \exists e \in A : r_t < \beta(e) \leq 2nr_t\,, \\ \min\{\beta(e) : e \in A, \beta(e) > r_t\}\,, & \text{otherwise.} \end{cases}$$

The sequence stops with $t = T$ once $r_t \geq \max_{e \in A} \beta(e)$. Note that $r_t \geq 2nr_{t-1}$ for all $t \in [T]$. We say that t is *regular*, if $r_t = 2nr_{t-1}$, and t is *special*, if $r_t > 2nr_{t-1}$; this means that $r_t = \beta(e)$ for some arc $e \in A$, and there is no arc $f \in A$ with $r_{t-1} < \beta(f) < r_t$.

We set $(\hat{N}_1, \hat{A}_1) = (N, A)$ in the first iteration. For both regular and special iterations for $t \geq 2$, the subgraph $(\hat{N}_t, \hat{A}_t)$ is obtained from $(\hat{N}_{t-1}, \hat{A}_{t-1})$ by

contracting the strongly connected components of arcs with $c(e) \leq r_{t-1}$; note that we use the original cost function c for this choice. For obtaining the contraction $(\hat{N}_t, \hat{A}_t)$ from $(\hat{N}_{t-1}, \hat{A}_{t-1})$, it suffices to consider the arcs such that $\beta(e) = r_{t-1}$.

We next define the subset of active arcs $F_t \subseteq \hat{A}_t$. This will be different for regular and special iterations. If t is regular, then we let

$$F_t := \left\{ e \in \hat{A}_t : c(e) \leq r_{t-1} < \beta(e) \leq r_t \right\},$$

and if t is special, then

$$F_t := \left\{ e \in \hat{A}_t : c(e) < \beta(e) = r_t \right\}.$$

In both cases, the graph $(\hat{N}_t, F_t)$ is acyclic. Indeed, any cycle containing $e \in F_t$ must include an arc of cost $c(f) \geq \beta(e)$. For regular iterations, F_t contains no such arcs, as they would have $c(f) > r_{t-1}$. For special iterations, it would mean $c(f) \geq r_t = \beta(e)$, a contradiction again.

Recall that $\hat{N}_t(F_t)$ denotes the subset of nodes that have at least one incident arc in F_t. We construct a topological ordering of $(\hat{N}_t(F_t), F_t)$ as $\hat{N}_t(F_t) = \{v_1, v_2, \ldots, v_s\}$ such that if $(v_i, v_j) \in F_t$, then $i < j$ must hold. We now define

$$p_t(v_j) = \frac{j r_t}{3n^2}, \quad j \in [s], \tag{3.3}$$

and increase $\pi(v_j)$ by $p_t(v_j)$ for every $j \in [s]$.

We note that from $(\hat{N}_{t-1}, \hat{A}_{t-1})$, we can identify $(\hat{N}_t(F_t), F_t)$ and the ordering all at once, by running Strongly-Connected$(\hat{N}_{t-1}(A''_{t-1}), A''_{t-1})$, where $A''_{t-1} = \{e \in \hat{A}_{t-1} : \beta(e) = r_{t-1}\} \cup F_t$.

After the final iteration, we uncontract all contracted sets in the reverse order, and obtain the potential π in the original graph. This finishes the description of Rough-balance, the subroutine asserted in Lemma 3.3.

Proof of Lemma 3.3. We need to show that the potential π defines a $7n^2$-min-balanced potential, and that the entire procedure can be implemented in time $O(m \log^2 n)$.

We start with the proof of balancedness; the key will be the following statement.

Claim 3.1. *Consider any $e \in A$, and let $\tau \in [T]$ such that $r_{\tau-1} < \beta(e) \leq r_\tau$. Then,*

$$\frac{r_\tau}{6n^2} \leq c^\pi(e) \leq \frac{7r_\tau}{6}.$$

Proof. The arc e will be contracted in iteration $\tau + 1$. That is, $e \in \hat{A}_\tau$, but the two endpoints of e coincide in $\hat{N}_{\tau+1}$. Hence, the final value of $c^\pi(e)$ will be the one seen in iteration τ.

We analyze the contribution of each potential p_t, $t \in [\tau]$ to the reduced cost $c^\pi(e)$. First, we show the upper bound. By the definition of p_t, we can bound the potential change caused by p_t as $|c^{p_t}(e) - c(e)| \leq (n-1) r_t/(3n^2)$. Thus, we obtain

(3.4)

$$\begin{aligned} |c^\pi(e) - c(e)| &\leq \frac{n-1}{3n^2} \cdot \sum_{t=1}^{\tau} r_t \\ &\leq \frac{n-1}{3n^2} \cdot r_\tau \left(1 + \frac{1}{2n} + \frac{1}{(2n)^2} + \ldots\right) \\ &= \frac{2(n-1)}{3(2n-1)n} \cdot r_\tau < \frac{r_\tau}{3n}, \end{aligned}$$

using that $r_t \geq 2n r_{t-1}$ for all $t \in [T]$. The upper bound follows by

$$c^\pi(e) \leq c(e) + \frac{r_\tau}{3n} \leq r_\tau + \frac{r_\tau}{3n} \leq \frac{7r_\tau}{6},$$

using that $n \geq 2$.

We distinguish two cases for the lower bound.

Case I: $e \in F_\tau$**.** By construction, $c^{p_\tau}(e) \geq c(e) + r_\tau/(3n^2)$ for all arcs in F_τ. As in (3.4), the potentials p_t for $t \in [\tau - 1]$ may decrease the reduced cost of e by at most $r_{\tau-1}/(3n) \leq r_\tau/(6n^2)$. The claimed bound follows.

Case II: $e \notin F_\tau$**.** First, we claim that $c(e) \geq r_\tau/(2n)$. Indeed, if t is a special value, then $e \notin F_\tau$ means $c(e) = \beta(e) = r_\tau$, and if t is a regular value, then $c(e) \geq r_{\tau-1} = r_\tau/(2n)$. Using (3.4), we see that

$$c^\pi(e) \geq c(e) - \frac{r_\tau}{3n} \geq \frac{r_\tau}{2n} - \frac{r_\tau}{3n} = \frac{r_\tau}{6n},$$

yielding a stronger lower bound.

Let us now show that c^π is $7n^2$-min-balanced. Consider any arc $e \in A$, and let $\tau \in [T]$ such that $r_{\tau-1} < \beta(e) \leq r_\tau$. Let $C \subseteq A$ be a cycle such that $e \in C$ and $c(f) \leq \beta(e)$ for all $f \in C$. Claim 3.1 implies that $c^\pi(e) \geq r_\tau/(6n^2)$, and at the same time, $c^\pi(f) \leq 7r_\tau/6$ for every $f \in C$. The desired balancedness property follows. ☐

Running time bound The initial call to Find-Balance(N, A, c) takes $O(m \log^2 n)$ time according to Lemma 3.4. The dominant terms in the running time bound are the calls to Strongly-Connected$(N_{t-1}(A''_{t-1}), A''_{t-1})$, where $A''_{t-1} = \{e \in \hat{A}_{t-1} : \beta(e) = r_{t-1}\} \cup F_t$. Let $m_{t-1} = |A''_{t-1}|$. We have $\sum_{i=0}^{T-1} m_t \leq 2m$, since every arc can participate in at most two sets A''_{t-1}. Hence, the calls to Strongly-Connected can be upper bounded as $O(\sum_{i=1}^{T-1} m_i) = O(m)$.

We do not need to maintain the entire contracted graph $(\hat{N}_t, \hat{A}_t)$, but only the arc sets F_t and A''_t, as well as the sets of incidents nodes N'_t and N''_t. These can be easily obtained by collecting all endnodes of the respective arc sets. The potential update in iteration t requires $O(|N'_t|)$. All these operations take altogether $O(m)$ time. The running time of the final uncontraction is $O(n)$ similarly to the argument in Section 3.2.

This subroutine can be implemented in the word RAM model. Whereas the potential values are not integers, they will always have the same denominator $3n^2$. Multiplying all costs initially by $3n^2$, we can work throughout with integer costs and potentials, and the algorithm only uses additions, comparisons, and multiplications/divisions by 2. Due to the assumption that n is a power of 2, the initial multiplication takes $O(1)$ time per arc. □

3.4 The strongly polynomial algorithm We now present the main algorithm as stated in Theorem 3.1. Since ξ-min-balancedness implies ξ'-min-balancedness for any $\xi' \geq \xi$, we can assume without loss of generality that $\lambda = 1 + 1/2^\rho$ for some $\rho \in \mathbb{Z}_{++}$. Replacing λ by $1 + 1/2^\rho$ such that $1 + 1/2^\rho \leq \lambda < 1 + 1/2^{\rho-1}$ does not affect the claimed asymptotic running time, and such a replacement is beneficial for the implementation in the RAM model. Thus, in this section we will assume $\lambda = 1 + 1/2^\rho$, and $\xi = 2\lambda - 1 = 1 + 1/2^{\rho-1}$ for some $\rho \in \mathbb{Z}_+$. Note that $\xi = 3$, $\lambda = 2$ corresponds to $\rho = 0$.

Given the initial input graph $G = (N, A, c)$ with integer arc costs, we first perform preprocessing to contract 0-cycles and to ensure that all arc costs are positive integers. Next, we apply the subroutine Rough-balance to find a $7n^2$-min-balanced reduced cost function. We assume that the input of Algorithm 3 is a positive and $7n^2$-min-balanced cost function c. If needed, we multiply all arc costs by $3n^2$ so that all reduced costs are integral. Recall that this takes only $O(1)$ time per arc due to the assumption that n is a power of 2.

Algorithm 3 is similar to the weakly polynomial Algorithm 1. The two crucial differences are that *(a)* Algorithm 3 may 'jump' over irrelevant values of L, and *(b)* the subroutine Small-Cycles is called only for a subset of 'active' arcs. A more detailed list of differences follows.

- The input digraph is required to be $7n^2$-balanced subgraph.
- At the beginning of the algorithm, we sort the arcs in the increasing order of costs $c(e)$.
- In iteration t, we denote the contracted image of the original graph and the relabelled cost as $(\hat{N}_t, \hat{A}_t, \hat{c}_t)$. However, we only maintain the $\hat{c}_t(e)$ values explicitly for a subset of *active arcs* $F_t \subseteq \hat{A}_t$. This is the set of arcs with $c(e) \leq (n+1)\lambda L_t$; we emphasize that the definition uses the input cost of e (for Algorithm 3) and not the relabelled cost.
- If an arc e first becomes active at iteration t, the subroutine Get-Cost(e) obtains the reduced cost $\hat{c}_t(e)$. For all other arcs, $\hat{c}_t(e)$ is defined, but not explicitly maintained in the algorithm.
- We call the subroutine Small-Cycles for the subgraph $(\hat{N}_t(F_t), F_t)$. This determines the node potential p_t and the partition $\mathcal{P}_t$ for contraction. With a slight abuse of notation, the node potentials p_t are extended to the entire node set $\hat{N}_t$, by setting $p_t(v) = 0$ for $v \in \hat{N}_t \setminus \hat{N}_t(F_t)$. Similarly, in line 7, the costs $\hat{c}_{t+1}$ and $\hat{c}_t^{p_t}$ are only updated for the arc set $\hat{F}$, the contracted image of F_t.
- Algorithm 1 maintains $D_t = (\lambda - 1)L_t$ in every iteration t. In contrast, in Algorithm 3 we set $D_t = L_t/2^\rho = (\lambda - 1)L_t$ whenever $t - 1$ is an integer multiple of 2^ρ, and maintain the same D_t value over the subsequent 2^ρ iterations. This is for the sake of staying within the word RAM model; in a more permissive arithmetic model, we could maintain $D_t = (\lambda - 1)L_t$ in every iteration. Note that for $\rho = 0$ and $\lambda = 2$, these two choices coincide.
- The choice of L_{t+1} again depends on the value of t. If t is not an integer multiple of 2^ρ, then we simply choose $L_{t+1} = L_t + D_t$. If t is an integer multiple of 2^ρ, then we use the selection in line 9. Our goal with this selection is to maintain $\hat{c}_{t+1}(e) \geq L_{t+1}$ for all $e \in \hat{A}_t$; Lemma 3.5 below shows that $\hat{c}_{t+1}(e) \geq c(e) - n\lambda L_t$.
- In line 13, we define F_{t+1} by taking the contracted image $\hat{F}$ of F_t, and adding all further arcs with $c(e) \leq (n+1)\lambda L_{t+1}$. This can be implemented straightforwardly using the initial sorting of the arc costs. For all new arcs, Get-Cost(e) in line 14 computes $\hat{c}_{t+1}(e)$.

Lemma 3.5. *Let $\tau \in [T]$ be an iteration such that in all previous iterations $t \in [\tau]$, $\hat{c}_t(e) \geq L_t$ was valid for all $e \in F_t$. Then, $|\hat{c}_{\tau+1}(e) - c(e)| \leq n\lambda L_\tau$ for every $e \in \hat{A}_t$.*

Proof. The condition guarrantees that the input to Small-Cycles at all iterations $t \leq \tau$ satisfied the requirement on the arc costs. The potential p_t found by Small-Cycles has values $-|\hat{N}_t|D_t \leq p_t(v) \leq 0$. Therefore, for each $e \in \hat{A}_t$, $|\hat{c}_{\tau+1}(e) - c(e)| \leq n \sum_{t=1}^{\tau} D_t$.

Algorithm 3 MIN-BALANCE

Input: A strongly connected directed graph $G = (N, A, c)$ with a $7n^2$-balanced cost vector $c \in \mathbb{Z}^A_{++}$, $\rho \in \mathbb{Z}_+$, such that $\lambda = 1 + 1/2^\rho$ and $\xi = 1 + 1/2^{\rho-1}$.

Output: A potential $\pi \in \mathbb{Q}^N$ such that c^π is ξ-min-balanced.

Sort all arcs in the increasing order of costs as $c(e_1) \le c(e_2) \le \ldots \le c(e_m)$;
$(\hat{N}_1, \hat{A}_1, \hat{c}_1) \leftarrow (N, A, c)$; $t \leftarrow 1$;
$L_1 \leftarrow \lfloor c(e_1) \rfloor_2$, $D_1 \leftarrow L_1/2^\rho$;
$F_1 \leftarrow \{e \in A : c(e) \le (n+1)\lambda L_1\}$;
while $|\hat{N}_t| > 1$ **do**
  $(\mathcal{P}_t, p_t) \leftarrow$ SMALL-CYCLES$(L_t, D_t, \hat{N}_t(F_t), F_t, \hat{c}_t)$;
  $(\hat{N}_{t+1}, \hat{F}, \hat{c}_{t+1}) \leftarrow (\hat{N}_t, F_t, \hat{c}_t^{p_t})/\mathcal{P}_t$;
  if t is an integer multiple of 2^ρ **then**
    $L_{t+1} \leftarrow \max\left\{L_t + D_t, \left\lfloor \min_{e \in \hat{A}_{t+1}} c(e) - n\lambda L_t \right\rfloor_2\right\}$;
    $D_{t+1} \leftarrow L_{t+1}/2^\rho$;
  else
    $L_{t+1} \leftarrow L_t + D_t$; $D_{t+1} \leftarrow D_t$;
  $F_{t+1} \leftarrow \hat{F} \cup \{e \in \hat{A}_{t+1} : (n+1)\lambda L_t < c(e) \le (n+1)\lambda L_{t+1}\}$;
  for $e \in F_{t+1} \setminus F_t$ **do** $\hat{c}_{t+1}(e) \leftarrow$GET-COST$(e)$;
Uncontract $(\hat{N}_t, \hat{A}_t, \hat{c}_t)$, and compute the overall potential $\pi : N \to \mathbb{Q}$.
return π.

We show that $\sum_{t=1}^{\tau} D_t \le \lambda L_\tau$. Indeed, $L_{t+1} \ge L_t + D_t$ in every iteration, implying $\sum_{t=1}^{\tau-1} D_t \le L_\tau$; and $D_\tau \le L_\tau/2^\rho = (\lambda - 1)L_\tau$. ∎

LEMMA 3.6. *For every iteration $t \in [T]$ in Algorithm 3, $\hat{c}_t(e) \ge L_t$ for all $e \in \hat{A}_t$. The final reduced cost function c^π is ξ-min-balanced. Further, every arc $e \in A$ with $c(e) < L_t/(7n^3)$ was contracted before iteration t.*

Proof. Let us start with the first claim. The proof is by induction. For $t = 1$, $\hat{c}_1(e) \ge L_1$ is true for every $e \in A = \hat{A}_1$ by the definition of $L_1 = \lfloor c(e_1) \rfloor_2$. Assume the claim was true for all $1 \le t' \le t$; we show it for $t+1$.

If we selected the value $L_{t+1} = \lfloor \min_{e \in \hat{A}_{t+1}} c(e) - n\lambda L_t \rfloor_2$, then the statement follows from Lemma 3.5. Let us next assume we selected $L_{t+1} = L_t + D_t$. If $e \in \hat{F}$, i.e., the contracted image of F_t, then $\hat{c}_{t+1}(e) \ge L_t + D_t = L_{t+1}$ is guaranteed by SMALL-CYCLES. Let $e \in \hat{A}_{t+1} \setminus F_t$, i.e., $c(e) > (n+1)\lambda L_t$. Then, Lemma 3.5 shows that $\hat{c}_{t+1}(e) > \lambda L_t \ge L_{t+1}$.

The ξ-min-balancedness property of the final reduced cost c^π follows as in Lemma 3.2 for the weakly polynomial Algorithm 1.

Consider now an arc $e \in A$ with $c(e) < L_t/(7n^3)$. By the $7n^2$-min-balancedness of the input cost function c, there exists a cycle $C \subseteq A$ such that $c(f) \le 7n^2 c(e)$ for all $f \in C$. The final reduced cost c^π is nonnegative, and therefore

$$c^\pi(e) \le c^\pi(C) = c(C) \le 7n^3 c(e) < L_t \,.$$

Recall that the final reduced cost $c^\pi(e)$ equals $\hat{c}_{t'}(e)$ for the iteration t' when f was contracted. Since $\hat{c}_t(f) \ge L_t$ for all $f \in \hat{A}_t$, it follows that $t' < t$, as required. ∎

In the full version we show that the overall running time of the operations GET-COST(e) can be bounded as $O(m\alpha(m, n))$ for the inverse Ackermann function $\alpha(m, n)$. We need one more claim that shows the geometric increase of L_t.

LEMMA 3.7. *For any iteration $t' \ge 1$, we have $L_{t'+2^\rho} \ge 2L_{t'}$.*

Proof. Let $t = t' + 2^\rho$. Assume first $2^\rho | t' - 1$. Then, $D_{t'} = L_{t'}/2^\rho$, and we have $D_{t''} = D_{t'}$ for all $t'' \in [t', t-1]$. Consequently, $L_t \ge L_{t'} + 2^\rho D_{t'} = 2L_{t'}$. The inequality may be strict if in iteration $t-1$ we defined L_t as the second term in (9).

Assume now $t' = t_0 + k$ such that $2^\rho | t_0 - 1$ and $k \in [1, 2^\rho - 1]$. Then, $L_{t'} = L_{t_0}(1 + k/2^\rho)$, $L_{t_0+2^\rho} \ge 2L_{t_0}$, and $L_t = L_{t_0+2^\rho}(1 + k/2^\rho) \ge 2L_{t_0}(1 + k/2^\rho)$, thus, we again have $L_t \ge 2L_{t'}$. ∎

We are now ready to prove Theorem 3.1.

Proof of Theorem 3.1. **The running time bound** We start by contracting all strongly connected components of 0-cost arcs in time $O(m)$, and run the algorithm ROUGH-BALANCE to find a $7n^2$-balanced cost function in time $O(m \log^2 n)$ (Lemma 3.3). We now turn the analysis of Algorithm 3. Let $m_t = |F_t|$ denote the number of active arcs in iteration t. The running

time of SMALL-CYCLES in iteration t is bounded as $\min\{O(1), O(m_t\sqrt{n})\}$. The term $O(1)$ is needed since there may be some 'idle' iterations without any active arcs, that is, $m_t = 0$. In such a case, within the next 2^ρ iterations L_t will be updated as in line 9, enforcing new active arcs. Thus, the number 'idle' iterations without active arcs can be bounded as $m2^\rho$, since every arc can give the minimum value in line 9 at most once. The total running time of the 'idle' iterations can be bounded as $O(m/(\lambda-1))$; this will be dominated by the other terms.

Let us now focus on the iterations containing active arcs. We show the following bound:

$$\sum_{t=1}^{T} m_t = O\left(\frac{m \log n}{\lambda - 1}\right). \tag{3.5}$$

Consider any arc $e \in A$. Let t_1 be the first and t_2 be the last iteration such that $e \in F_t$. By definition, t_1 is the smallest value such that $c(e) \leq (n+1)\lambda L_{t_1}$, and by the last part of Lemma 3.6 $L_{t_2}/(7n^3) \leq c(e)$. Thus, $L_{t_2} \leq 14n^4 L_{t_1}$. Lemma 3.7 shows that L_t increases by a factor 2 in every $2^\rho = 1/(\lambda-1)$ iterations. Hence, $t_2 - t_1 \leq \frac{\log(14n^4)}{\lambda-1}$, implying (3.5).

Hence, the total running time of the calls to SMALL-CYCLES is bounded as $O\left(\frac{m\sqrt{n}\log n}{\lambda-1}\right)$. The time of contractions and cost updates can be bounded as $O(m\alpha(m,n))$ using the Union-Find-Increase data structure, as shown in the full version. The final uncontraction takes $O(n)$. The overall running time bound follows.

Implementation in the word RAM model The subroutines ROUGH-BALANCE and SMALL-CYCLES are both implementable in the word RAM model. Algorithm 3 only uses operations permitted in the model: we only multiply by factors of 2, and $\lfloor . \rfloor_2$ can be implemented by bit-shifting. We also note that L_t is a power of 2 in every iteration when $2^\rho | t-1$.

Let us further multiply all the input arc costs such that they become integer multiples of 2^ρ by a bit-shift operation. For this modified input, D_t is also an integer multiple of 2 in every iteration, and therefore the potential changes and all reduced costs remain integers throughout.

Obtaining the component hierarchy Assume now that $\lambda = 2$, and we use the algorithm as described in Algorithm 3. The sets contracted during the algorithm can be naturally represented by a rooted tree $(V \cup N, E)$, where the nodes N correspond to the leaves and the root $r \in V$ to the final contraction of the entire node set. If the set represented by some $v \in V$ was contracted at iteration t, we set $a(v) = L_t$.

We claim that $(V \cup N, E, a)$ forms a component hierarchy of $G^\pi = (N, A, c^\pi)$ with parameter $\lambda = 2$. All $a(v) = L_t$ values are integer powers of 2. It is immediate that the leaves in the subtree of each $v \in V$ form a strongly connected component in G^π. Let v represent a set contracted in iteration t, that is, $v = P_i$ for a set P_i in the partition $\mathcal{P}_t$. If $\text{lca}(i,j) = v$ for $i, j \in N$, that means that the nodes i and j got contracted together in iteration t. We show that $a(v) \leq \beta(i,j) \leq \xi a(v)$, and that the nodes of $\text{desc}(v)$ contains a bottleneck shortest path between i and j. If $t = 1$, then $L_1 = \lfloor C_{\min} \rfloor_2$, and P_i is strongly connected in the subgraph of arcs of cost at most ξL_1. If $t > 1$, then $(\hat{N}_t, \hat{A}_t)$ contains a path between the contracted images of i and j with all arc costs between $a(v) = L_t$ and ξL_t, and every i–j path must contain an arc of cost $\geq L_t$. We can map this back to the original graph by uncontracting the sets from previous iterations; all arc obtained in the uncontraction will have costs $\leq \xi L_{t-1} < \xi L_t$. □

4 The shortest path algorithm

In this section, we assume that a 3-min-balanced directed graph $G = (N, A, c)$ is given as is a component hierarchy $(V \cup N, E, r, a)$ for G with parameter 2. Theorem 3.1 shows that the time to obtain reduced cost c^π such that $G^\pi = (N, A, c^\pi)$ is 3-min-balanced, and the time to obtain a corresponding component hierarchy for G^π is $O(m\sqrt{n}\log n)$. Further, recall that for any i–j path P, $c^\pi(P) = c(P) - \pi(i) + \pi(j)$. Therefore, the set of shortest paths between any two nodes is the same in G and G^π. For simplicity of notation, in this section we assume that the input cost function c is already 3-min-balanced, integral, and strictly positive; we can achieve integrality by appropriately multiplying the cost function. The algorithm described in this section is an adaptation of Thorup's [26] result to the setting of balanced directed graphs.

4.1 Upper bounds for the component hierarchy

Let us be given a component hierarchy $(V \cup N, E, r, a)$ for a directed graph $G = (N, A, c)$ with parameter $\lambda > 1$. For a node $u \in V \cup N$, the *height* $h(u)$ is the length of the longest path between u and a node in $\text{desc}(u)$; each $u \in N$, we have $h(u) = 0$. We define the functions $U, \eta : V \to \mathbb{Q}$ recursively, in non-decreasing order of $h(u)$ as

$$\begin{aligned} U(v) :=& (2\lambda - 1)a(v)(|\text{ children}(v)| - 1) \\ &+ \sum_{v' \in \text{children}(v) \setminus N} U(v'), \\ \eta(v) :=& 1 + \left\lceil \frac{U(v)}{a(v)} \right\rceil. \end{aligned} \tag{4.6}$$

LEMMA 4.1. *Let $(V \cup N, E, r, a)$ be a component hierarchy for a directed graph $G = (N, A, c)$ with parameter*

λ, and let U, η be as in (4.6). For any pair of nodes $i, j \in N$ and $v = \mathrm{lca}(i, j)$, there is an i–j path P of length at most $U(v)$, where each node of P is in $\mathrm{desc}(v)$*. In addition,*

$$\sum_{v \in V} \eta(v) < (2\lambda + 2)|N|\,.$$

Proof. Let $i, j \in N$ and $v = \mathrm{lca}(i, j)$. The proof is by induction on $h(v)$. Consider the i–j path P' in $\mathrm{desc}(v)$ such that $c(e) \le (2\lambda - 1)a(v)$ for all $e \in P'$, as guaranteed by the property of the component hierarchy.

In the base case $h(v) = 1$, the bound is immediate, since P' has at most $|\,\mathrm{children}(v)| - 1$ arcs. Assume now $h(v) > 1$, and that the statement holds for any i', j' with $h(\mathrm{lca}(i', j')) < h(v)$. One can choose an i–j path P that satisfies the following property for each child u of v. If i' and j' are the first and last nodes of P that are in $\mathrm{desc}(u)$, then the subpath in P from i' to j' consists of nodes of $\mathrm{desc}(u)$. By the inductive hypothesis, for each child u of v, the length of the subpath in $\mathrm{desc}(u)$ is at most $U(u)$. There are at most $|\,\mathrm{children}(v)| - 1$ arcs in P between different $\mathrm{desc}(u)$ subpaths; their cost is at most $(2\lambda - 1)a(v)(|\,\mathrm{children}(v)| - 1)$. Thus, the bound $c(P) \le U(v)$ follows.

Let us now turn to the second statement. We analyze the contribution of each $i \in N$ to the sum $\sum_{v \in V} U(v)/a(v)$. Let $i = v_0, v_1, v_2, \ldots, v_k = r$ be the unique path in the tree (V, E) from i to the root; thus, $p(v_t) = v_{t+1}$ for $t = 0, \ldots, k-1$. Then, the contribution of i to each $U(v_t)$ is less than $(2\lambda - 1)a(v_1)$. Using that $a(v_{t+1}) \ge \lambda a(v_t)$ for each $t = 0, \ldots, k-1$, we see that

$$\sum_{v \in V \setminus N} \frac{U(v)}{a(v)} < (2\lambda - 1) \sum_{i \in N} \sum_{t=1}^{\infty} \frac{1}{\lambda^{t-1}} = \frac{(2\lambda - 1)\lambda}{\lambda - 1}|N| < 2\lambda|N|\,.$$

The statement follows noting also that $|V \setminus N| \le |N| - 1$, since $(V \cup N, E)$ is a tree, and $\eta(v) < 2 + (U(v)/a(v))$ for all $v \in V \setminus N$. ∎

4.2 Setting the stage Given the input directed graph $G = (N, A, c)$, our goal is to compute the shortest path distances from a *source node* $s \in N$ to all nodes in N. We will assume that c is a 3-balanced integer cost function, and that we are given the component hierarchy $(V \cup N, E, r, a)$ for G with parameter 2, obtained as in Theorem 3.1. Further, the bounds $U, \eta : V \to \mathbb{Q}_+$ are defined as in (4.6).

We now summarize the notation used in the algorithm; for convenience, we repeat some notation from Section 2.

Pointers in the component hierarchy

$\mathrm{p}(v)$	the parent of a node/vertex $v \in (V \cup N) \setminus \{r\}$
$\mathrm{children}(v)$	the set of nodes/vertices u such that $\mathrm{p}(u) = v$
$\mathrm{desc}(v)$	the set of nodes $i \in N$ that are leaves in the subtree rooted at $v \in V$
$\mathrm{lca}(u, v)$	the least common ancestor of nodes/vertices u and v

Distances labels for nodes and vertices

$d(i)$: $i \in N$	the shortest distance from the source s to node $i \in N$
$d(v)$: $v \in V$	defined as $\min\{d(i) : i \in \mathrm{desc}(v)\}$
$D(i)$: $i \in N$	an upper bound on $d(i)$ as computed by the algorithm
$D(v)$: $v \in V$	an upper bound on $d(v)$, to be specified later
$S \subseteq N \cup V$	a set of *permanent* nodes and vertices in the algorithm
$\mathrm{pred}(i)$	the predecessor of node i in the algorithm

If $i \in S$ then $D(i) = d(i)$. Nodes in $N \setminus S$ are called *temporary*. The values $D(i)$ are initialized as $D(s) = 0$ and $D(i) = \infty$ for $i \ne s$. The predecessors are defined for nodes with finite $D(i)$ values. The bound $D(i)$ is propagated from the in-neighbour $\mathrm{pred}(i) \in S$, that is, $D(i) = D(\mathrm{pred}(i)) + c(\mathrm{pred}(i), i)$ for $i \in N \setminus \{s\}$. The graph of the arcs $(\mathrm{pred}(i), i)$ is acylic, and contains a path from the source s to every node $i \in N$ with $D(i) < \infty$.

Active vertices and buckets Each vertex $v \in V$ can be *active* or *inactive*. Initially, the root r is the only active vertex. A vertex can only become active if all its ancestors are active. Once activated, a vertex remains active until it becomes permanent (enters S). One of the active vertices will be CV, the *current vertex*, initalized as $\mathrm{CV} = r$. When a vertex is activated, we create a data structure of buckets.

$\mathrm{Buckets}(v)$	the list of buckets for vertex $v \in V$
$\mathrm{FirstB}(v)$	the first bucket of the list $\mathrm{Buckets}(v)$
$\mathrm{LastB}(v)$	the last bucket of $\mathrm{Buckets}(v)$
$\mathrm{Upper}(v)$	the upper bound associated with $\mathrm{LastB}(v)$
$\mathrm{CurrentB}(v)$	the "current bucket" of vertex v
$\mathrm{NextB}(v)$	the bucket that follows $\mathrm{CurrentB}(v)$; $\mathrm{NextB}(\mathrm{LastB}(v)) = \emptyset$
$L(B)$	the lower bound associated with bucket B

For the kth bucket $B \in \text{Buckets}(v)$, $L(B)$ is an integer multiple of $a(v)$. A child $i \in \text{children}(v)$ is stored in bucket $B \in \text{Buckets}(v)$ if $L(B) \le D(i) < L(B)+a(v)$.

A node i is contained in a bucket of their parent $p(i)$ if and only if $p(i)$ is active. Otherwise, we maintain a pointer $\text{GIA}(i)$ that denotes the *greatest inactive ancestor* of i. This value is maintained using the Split/FindMin data structure introduced by Gabow [10], as described in the full version.

Distances labels for vertices For vertices $v \in V$, the distance label $D(v)$ is defined as follows:

- If v is active, then $D(v) = L(\text{CurrentB}(v))$.
- If v is inactive, and $p(v)$ is active, then $D(v) = \min\{D(i) : i \in \text{desc}(v)\}$.
- If both v and $p(v)$ are inactive, then $D(v)$ is undefined.

For inactive vertices, $D(v)$ is also maintained using the Split/FindMin data structure.

Algorithm 4 SHORTEST-PATHS

Input: A directed graph $G = (N, A, c)$ with $c \in \mathbb{Z}^A_{++}$, source node $s \in N$, a component hierarchy $(V \cup N, E, a)$ for G with parameter 2.

Output: Shortest path labels for each $i \in N$ from s.

1: $S \leftarrow \emptyset$;
2: $D(s) \leftarrow 0$; $D(r) \leftarrow 0$;
3: **for** $j \in N \setminus \{s\}$ **do** $D(j) \leftarrow \infty$;
4: **for** $v \in V$ **do** compute $U(v)$ and $\eta(v)$ as in (4.6) ;
5: ACTIVATE(r) ;
6: CV $\leftarrow r$;
7: **while** $D(r) < \text{Upper}(r)$ **do** MAIN ;
8: **return** labels $D(i)$: $i \in N$.

4.3 Description of the algorithm The algorithm is a bucket-based *label setting* algorithm, similarly to a bucket-based implementation of Dijkstra's algorithm. For each node $i \in N$, we maintain an upper bound $D(i)$ on the true distance $d(i)$ from s, and gradually extend the set S of permanent nodes. A node i enters S when the following are true: the current vertex is $v = p(i)$, and the algorithm selects node i from the bucket $\text{CurrentB}(v)$. At the iteration at which i enters S, $D(i) = d(i)$. For nodes $i \in N \setminus S$, $D(i)$ is the length of the shortest s–i path in $S \cup \{i\}$; there is such a path if and only if $D(i) < \infty$.

Dijkstra's algorithm always adds a node $j \in N \setminus S$ to S that has minimal $D(j)$ value. Our algorithm may add nodes whose labels are non-minimal, but we maintain the property that when j is added,

$$D(j) \le D(i) + b(i,j) \quad \forall j \in N \setminus S\,. \tag{4.7}$$

This condition is sufficient for establishing the correctness of the algorithm. (See Lemma 4.4).

The overall algorithm is shown in Algorithm 4. After initialization, it repeatedly calls the subroutine MAIN (Algorithm 7), which relies on the subroutines ACTIVATE(v) (Algorithm 5) and UPDATE(i) (Algorithm 6). The subroutines rely on the Split/FindMin data structure, as detailed in the full version.

Initially, the current vertex is set as the root: CV $= r$. We now outline the different cases in MAIN. At the current vertex, $B = \text{CurrentB}(\text{CV})$ is the smallest nonempty bucket.

If B contains a node i, we make it permanent, i.e., add it to S. When a node enters S, the subroutine UPDATE(i) updates the labels for each out-neighbour j of i to $\min\{D(j), D(i) + c(i,j)\}$, similarly to Dijkstra's algorithm. We then update the estimates $D(v)$ for vertices. Namely, if $D(j)$ decreases, then we may need to update $D(\text{GIA}(j))$ using Split/FindMin. The node i is then removed from the bucket B.

If the bucket B does not contain any nodes, but contains some vertices, then we set the new current vertex as one of the children v of the current vertex CV in this bucket. If v is inactive, we call ACTIVATE(v). This subroutine creates the bucket data structure at v, and updates the $D(w)$ values of the child vertices w of v using Split/FindMin.

The subroutine ACTIVATE(v) requires the following procedures. ADDBUCKET(v) creates an empty bucket and appends it to the end of the list Buckets(v). We define Upper(v) as the upper bound associated with the last bucket of v.

The procedure MOVETOBUCKET(j) first lets $v = p(j)$, then determines if there is a bucket B' of Buckets(v) whose range contains the value $D(j)$. This can be identified by the division $\lfloor D(j)/a(v) \rfloor$; recall that $a(v)$ is a power of two, and thus this can be obtained by bit-shift operations. If there is no such bucket, that is, $D(j) \notin [L(v), \text{Upper}(v)]$, no operation is performed. Otherwise, j is added to the bucket B' unless j was already contained in B'. If j was already contained in some other bucket $B'' \in \text{Buckets}(v)$, then j is deleted from B''. The procedure DELETEFROMBUCKET(j) deletes the node/vertex j from the bucket of Buckets($p(j)$).

The remaining possibility in MAIN is when the bucket B becomes empty in the current iteration. We update the label $D(\text{CV})$ to $D(\text{CV}) + a(\text{CV})$. If B is not already the last bucket at CV, then we move the

current bucket of CV to NextB(B). If CV $\neq r$, we check if the increase in D(CV) causes CV to move up to the next bucket of p(CV). In such a case, we change the current vertex CV to its parent p(CV). Finally, in case B = LastB(CV), we are finished processing CV. We add this vertex to S, and the current vertex is moved to p(CV), unless CV $= r$. The algorithm terminates once the final bucket LastB(r) becomes empty.

Note that in the first iterations, the current vertex moves down along the r–s path. Once CV $= p(s)$, the first node added to S is s.

Let us now sketch the reason why the property (4.7) holds when j is added to S. This can happen if j is in the current bucket B of the current vertex CV $= p(i)$. Consider any $i \in N \setminus (S \cup \{j\})$, and let $v = \mathrm{lca}(i, j)$. Then, $b(i, j) \geq a(v)$ by the property of the component hierarchy; we show the stronger property $D(j) < D(i) + a(v)$ that implies (4.7). This stronger property is a consequence of the following two properties: $D(v) \leq D(i)$ (Lemma 4.4), and $D(j) < D(v) + a(v)$ (Lemma 4.5).

Algorithm 5 The ACTIVATE subroutine

```
1: procedure ACTIVATE(v)
2:     Buckets(v) ← 0 ;
3:     B ← ADDBUCKET(v) ;
4:     FirstB(v) ← B ; CurrentB(v) ← B ;
5:     Λ ← a(v) ⌊D(v)/a(v)⌋ ;
6:     L(B) ← Λ ;
7:     for k = 2, ..., η(v) do
8:         Λ ← Λ + a(v) ;
9:         B ← ADDBUCKET(v) ;
10:        L(B) ← Λ ;
11:    Upper(v) ← Λ + a(v) ;
12:    for w ∈ children(v) ∩ V do
13:        D(w) ← min{D(i) : i ∈ desc(v)} ;   ▷ using the Split/FindMin data structure
14:        MOVETOBUCKET(w) ;
15:    for w ∈ children(v) ∩ N do
16:        MOVETOBUCKET(j) ;
```

4.4 Analysis

THEOREM 4.1. *For $G = (V, E, c)$ with $c \in \mathbb{Z}^E_{++}$ with $n = |N|$ and $m = |A|$, and provided the component hierarchy, Algorithm 4 computes shortest paths from node $s \in N$ to all other nodes in $O(m)$ time.*

We prove the theorem as a sequence of the following lemmas. The first two lemmas show the running time bound, and the rest shows correctness, namely, that the algorithm adds every node in N to S and has label $D(i) = d(i)$ when i is added to S.

Algorithm 6 The UPDATE subroutine

```
1: procedure UPDATE(i)
2:     for (i, j) ∈ A(i) do
3:         if D(i) + c(i, j) < D(j) then
4:             D(j) ← D(i) + c(i, j) ;
5:             pred(j) ← i ;
6:             if p(j) is active then MOVETOBUCKET(j) ;
7:             else
8:                 w ← GIA(j) ;
9:                 D(w) ← min{D(j), D(w)} ;   ▷ using the Split/FindMin data structure
```

LEMMA 4.2. *The Procedure* MAIN() *is called is $O(n)$ times.*

Proof. Let B be the current bucket at an iteration of MAIN(). We consider the cases according as *(i)* B contains a node, or *(ii)* B contains a vertex but no node, or *(iii)* B is empty.

Whenever case *(i)* occurs, a node is added to S, giving a bound of $O(n)$ for this case. In case *(iii)*, CurrentB(v) is replaced by NextB(B), and CV is possibly replaced by p(CV). The number of times this can occur is equal to the total number of buckets, which is $O(n)$ by Lemma 4.1.

Let us now turn to case *(ii)*. Let τ denote the distance of vertex CV from the root r in the component hierarchy. Both in the first and the final iteration, CV $= r$, and thus $\tau = 0$. Whenever case *(ii)* occurs, τ increases by one. The only way τ can decrease is in case *(iii)*, if the current bucket of CV is emptied and CV is replaced by p(CV), unless CV $= r$ and we terminate. Thus, the total number of occurrences of case *(ii)* is equal to the total number of increases in τ, which equals the total number of decreases, which is bounded by $O(n)$. Thus, each of the three cases can only occur $O(n)$ times, and the claim follows. ∎

LEMMA 4.3. *The total running time of Algorithm 4 is bounded as $O(m)$.*

Proof. The time for initialization is $O(n)$. We now consider the time for MOVETOBUCKET. Each call takes $O(1)$ time, and it is called once each time that $D(i)$ is decreased for $i \in N$. (If $p(i)$ is active, then node i is placed in a bucket. If $p(i)$ is inactive, then GIA(i) is placed in a bucket.) Thus, the total time for MOVETOBUCKET is $O(m)$.

We now consider UPDATE(i). At each call, the arcs in $A(i)$ are scanned. The time to update $D(j)$ for $(i, j) \in$

Algorithm 7 The MAIN subroutine

```
1: procedure MAIN
2:     v ← CV ;
3:     B ← CurrentB(v) ;
4:     if B ∩ N ≠ ∅ then
5:         select a node i ∈ B ∩ N ;
6:         DELETEFROMBUCKET(i) ;
7:         S ← S ∪ {i} ;
8:         UPDATE(i) ;
9:     else if B ∩ V ≠ ∅ then
10:        select a vertex w ∈ B ∩ V ;
11:        CV ← w ;
12:        if w is inactive then ACTIVATE(w) ;
13:    else
14:        D(v) ← D(v) + a(v) ;
15:        if D(v) = Upper(v) and v ≠ r then
16:            S ← S ∪ {v} ;
17:            delete v from its bucket in Buckets(p(v))
    ;
18:            CV ← p(v) ;
19:        if D(v) < Upper(v) then
20:            CurrentB(v) ← NextB(B)
21:            if v ≠ r and D(v) ≥ D(p(v)) + a(p(v))
    then
22:                delete v from its bucket in
    Buckets(p(v)) ;
23:                move v to the bucket
    NextB(CurrentB(p(v))) ;
24:                CV ← p(v) ;
```

$A(i)$ is $O(1)$. If $p(j)$ is active, then the time to put node j in the correct bucket of Buckets$(p(j))$ is $O(1)$. A potential bottleneck occurs when $p(j)$ is inactive and $D(j)$ is updated. In this case, the algorithm determines $w = \text{GIA}(j)$ and then updates $D(w)$. The amortized time to determine w and update $D(w)$ is $O(1)$ using Thorups [26] implementation of the Split/FindMin data structure.

We now consider the time for ACTIVATE(v). This procedure is called $O(n)$ times, and the total number of buckets is $O(n)$. The potential bottleneck is updating $D(w)$ for $w \in V$. This is accomplished using the Split/FindMin data structure. Accordingly, the running time is the same as for the UPDATE step.

Finally, we consider the time spent in Procedure Main. By Lemma 4.2, this procedure is called $O(n)$ times in total. Not including the time for the subroutines UPDATE and ACTIVATE, the running time is $O(n)$ in total. This completes the proof of the lemma. ∎

LEMMA 4.4. *Suppose that vertex w is a descendant of vertex v, and that both w and v are active. Then, $D(v) \le D(w) \le D(v) + a(v)$.*

The proof is given in the full version of the paper, using induction on the length of the unique path from w to v in the component hierarchy.

LEMMA 4.5. *Let $j \in N \setminus S$ and let v be an active ancestor of j. Then, $D(v) \le D(j)$. In the iteration when j is added to S, we also have $D(j) < D(v) + a(v)$.*

Proof. For every $j \in N \setminus S$, let $\bar{p}(j) = p(j)$ if $p(j)$ is active, and $\bar{p}(j) = \text{GIA}(j)$ otherwise. That is, $\bar{p}(j)$ is the nearest ancestor of j for which $D(\,)$ is defined. For the first claim of the lemma, it suffices to show that for $v = \bar{p}(j)$, $D(v) \le D(j)$. Then, $D(v') \le D(j)$ follows for every ancestor v' of j. If v is active, then $D(v') \le D(v)$ for every ancestor v' of v by Lemma 4.4. If v is inactive, but $p(v)$ is active, then $D(p(v)) \le D(v)$ and $D(v') \le D(p(v))$ for every ancestor v' of $p(v)$. For the second claim of the lemma, it suffices to show that it is true when $v = p(j)$.

We establish the second claim now in the case that $v = p(j)$. When j is added into S, it is in the first bucket of v. Thus, $D(j) < D(v) + a(v)$.

We now establish the first claim of the lemma. If $\bar{p}(j) = \text{GIA}(j)$, we claim that $D(v) \le D(j)$. To see why, let $x = \text{GIA}(j)$. Then x is inactive, and $p(x)$ is active. Accordingly, the algorithm maintains that $D(x) = \min\{D(i) : i \in \text{desc}(x)\}$, and the claim is true.

For the case that $p(j)$ is active, we prove it using induction on the number of iterations. The claim is clearly true at the first iteration, where the only active vertex is r and $D(r) = 0$. We show that if the claim is true for every $j \in N \setminus S$ in iteration t, then it remains true in iteration $t + 1$.

The only case that needs to be considered is when the label of node j is decreased, which occurs when UPDATE(i) is executed for some $i \in N$ such that $(i, j) \in A$. Let $D(.)$ and $D'(.)$ denote the labels before and after this call to UPDATE(i), and let $z = \text{lca}(i, j)$, and $w = p(i)$. By induction, $D(w) \le D(i)$.

If $z = v$ (and thus w is a descendant of v), then Lemma 4.4 implies $D(v) \le D(w)$, and thus,

$$D'(j) = D(i) + c(i, j) \ge D(w) \ge D(v) = D'(v)\,.$$

Next, assume that $v \ne z$. Lemma 4.4 and the inductive hypothesis imply

$$D(v) - a(z) \le D(z) \le D(w) \le D(i),$$

and therefore,

$$D'(j) = D(i) + c(i, j) \ge D(v) + c(i, j) - a(z) \ge D(v)\,.$$

The final inequality follows since $b(i, j) \le c(i, j)$, and $a(v) \le b(i, j)$ is a required property of the component hierarchy. ∎

LEMMA 4.6. *For every vertex $v \in V$ and descendant $i \in \mathrm{desc}(v)$, $d(i) < \mathrm{Upper}(v)$.*

Proof. $\mathrm{Upper}(v)$ is computed in ACTIVATE(v). When this subroutine is called, $D(v)$ is the minimum of $D(i)$ over $i \in \mathrm{desc}(v)$. Let $i_0 \in \mathrm{desc}(v)$ be the node giving the minimum, that is, $D(v) = D(i_0)$. Then, for every $j \in \mathrm{desc}(v)$, Lemma 4.1 asserts that $d(j) \leq d(i_0) + U(v) \leq D(v) + U(v)$.

In line 5 of ACTIVATE(v), we set the value of Λ as $D(v) - a(v) < \Lambda \leq D(v)$, where $\Lambda = L(\mathrm{FirstB}(v))$. The algorithm then defines $\mathrm{Upper}(v) = \Lambda + \eta(v)a(v)$. Recall from (4.6) tha $\eta(v) = 1 + \lceil U(v)/a(v) \rceil$. Thus, $\mathrm{Upper}(v) > D(v) + U(v) \geq d(j)$ follows. ∎

LEMMA 4.7. *The algorithm terminates with $N \subseteq S$, and $D(j) = d(j)$ for every $j \in N$.*

Proof. Let j be any node of N. Let $P = i_1, i_2, \ldots, i_k$ (where $i_1 = s$ and $i_k = j$) be a shortest path from node s to node j. We claim that every node in P will be added to S, and they will be added to S in increasing order of index. In such a case, $D(j) = d(j)$ follows.

First, let us consider the case when every node in P was added to S during the algorithm. We prove by contradiction that the nodes are added to S in increasing order of the index.

Suppose that node i_p is the first index node selected prior to its predecessor i_{p-1}. Let i_ℓ be the last node of P selected prior to node i_p. Thus, $D(i_\ell) = d(i_\ell)$. Moreover, subsequent to carrying out UPDATE(i_ℓ), $D(i_{\ell+1}) = d(i_{\ell+1})$. Let P' be the subpath of P from $i_{\ell+1}$ to i_p. Let v be the lowest common ancestor of the nodes of P'. Thus, there is some arc e of P' with $c(e) \geq a(v)$.

By the second part of Lemma 4.5, at the iteration at which i_p was selected, the following is true:

(4.8)
$$\begin{aligned} D(i_p) &< D(v) + a(v) \leq D(i_{\ell+1}) + c(e) \\ &= d(i_{\ell+1}) + c(e) \leq d(i_{\ell+1}) + c(P') = d(i_p) \end{aligned}$$

But this is a contradiction, showing that if every node in P was added to S, then $D(j) = d(j)$ at termination.

Next, assume that some nodes in P are not included in S throughtout the algorithm; we show that this leads to a contradiction. Let i_p be the first node in P not added to S; clearly $p \geq 2$. The above argument shows that each of $i_1, i_2, \ldots, i_{p-1}$ were added to S in this order, and $D(i_k) = d(i_k)$ for all $k \in [p-1]$ at the time they were added to S. Further, UPDATE(i_{p-1}) has set $D(i_p) = d(i_p)$.

Consider the final iteration when the algorithm terminated at $D(r) = \mathrm{Upper}(r)$. Since the root r is an ancestor of i_p, $D(r) \leq D(i_p)$ according to Lemma 4.5. On the other hand, Lemma 4.6 shows $d(i_p) < \mathrm{Upper}(r)$. We obtain a contradiction from $D(i_p) = d(i_p)$. ∎

5 Conclusions

In this paper, we have given an $O(mn)$ algorithm for the directed all pairs shortest paths problem with nonnegative integer weights. Our algorithm first replaces the cost function by a reduced cost satisfying an approximate balancing property in $O(m\sqrt{n}\log n)$ time. Subsequently, every shortest path computation can be done in linear time, by adapting Thorup's algorithm [26].

One might wonder if our technique may also lead to an improvement for APSP in the addition-comparison model, where the best running time is $O(mn + n^2 \log\log n)$ by Pettie [21]. This running time bound is based on multiple bottlenecks. However, as explained in Section 1.1.1, the approximate cost balancing is able to get around the sorting bottleneck of [21]. Using the $O(m \log \alpha(n, m))$ implementation of Split/FindMin, an overall $O(mn \log \alpha(n, m))$ might be achievable.

However, there is one remaining important bottleneck where our algorithm crucially relies on bit-shift operations: the operation MOVETOBUCKET(j), which places a node/vertex in the bucket at $v = p(j)$ containing the value $D(j)$. Pettie and Ramachandran [22] show that these operations can be efficiently carried out in $O(1)$ amortized time per operation in a bucket-heap data structure, assuming the hierarchy satisfies certain *'balancedness'* property. Section 5 of the paper shows how the 'coarse hierarchy' obtainable from a minimum spanning tree and used by Thorup can be transformed to a 'balanced hierarchy'. This method does not seem to easily apply to the directed hierarchy concept used in this paper.

Our approximate min-balancing algorithm may be of interest on its own, and has strong connections to the matrix balancing literature as detailed in Section 1.1.2. For finding an $(1+\varepsilon)$-min-balanced reduced cost for $\varepsilon = O(1)$, our algorithm takes $O(\frac{m\sqrt{n}\log n}{\varepsilon})$ time. One might wonder if there is an algorithm with the same polynomial term $\tilde{O}(m\sqrt{n})$ but with a dependence on $\log(1/\varepsilon)$. We note that the algorithm in [24] for approximate max-balancing has a $\log(1/\varepsilon)$ dependence.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows – Theory, Algorithms and Applications*. Prentice Hall, 1993.

[2] Z. Allen-Zhu, Y. Li, R. Oliveira, and A. Wigderson. Much faster algorithms for matrix scaling. In *58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 890–901, 2017.

[3] J. M. Altschuler and P. A. Parrilo. Random Osborne: a simple, practical algorithm for matrix balancing in near-linear time. *arXiv preprint arXiv:2004.02837*, 2020.

[4] M. B. Cohen, A. Madry, D. Tsipras, and A. Vladu. Matrix scaling and balancing via box constrained Newton's method and interior point methods. In *58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 902–913, 2017.

[5] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[6] E. Dinic. Economical algorithms for finding shortest paths in a network. *Transportation Modeling Systems*, pages 36–44, 1978.

[7] R. W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[8] M. L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing*, 5(1):83–89, 1976.

[9] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[10] H. N. Gabow. A scaling algorithm for weighted matching on general graphs. In *26th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 90–100, 1985.

[11] A. V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, 1995.

[12] T. Hagerup. Improved shortest paths on the word RAM. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 61–72. Springer, 2000.

[13] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23(3):309–311, 1978.

[14] S. T. McCormick. Approximate binary search algorithms for mean cuts and cycles. *Operations Research Letters*, 14(3):129–132, 1993.

[15] J. B. Orlin and R. K. Ahuja. New scaling algorithms for the assignment and minimum mean cycle problems. *Mathematical Programming*, 54(1-3):41–56, 1992.

[16] J. B. Orlin and A. Sedeño Noda. An $o(nm)$ time algorithm for finding the min length directed cycle in a graph. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1866–1879, 2017.

[17] E. Osborne. On pre-conditioning of matrices. *Journal of the ACM (JACM)*, 7(4):338–345, 1960.

[18] R. Ostrovsky, Y. Rabani, and A. Yousefi. Matrix balancing in l_p norms: bounding the convergence rate of Osborne's iteration. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 154–169, 2017.

[19] B. N. Parlett and C. Reinsch. Balancing a matrix for calculation of eigenvalues and eigenvectors. *Numerische Mathematik*, 13(4):293–304, 1969.

[20] S. Pettie. On the comparison-addition complexity of all-pairs shortest paths. In *International Symposium on Algorithms and Computation*, pages 32–43. Springer, 2002.

[21] S. Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theoretical Computer Science*, 312(1):47–74, 2004.

[22] S. Pettie and V. Ramachandran. A shortest path algorithm for real-weighted undirected graphs. *SIAM Journal on Computing*, 34(6):1398–1431, 2005.

[23] H. Schneider and M. H. Schneider. Max-balancing weighted directed graphs and matrix scaling. *Mathematics of Operations Research*, 16(1):208–222, 1991.

[24] L. J. Schulman and A. Sinclair. Analysis of a classical matrix preconditioning algorithm. *Journal of the ACM (JACM)*, 64(2):1–23, 2017.

[25] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[26] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM (JACM)*, 46(3):362–394, 1999.

[27] M. Thorup. Integer priority queues with decrease key in constant time and the single source shortest paths problem. *Journal of Computer and System Sciences*, 69(3):330–353, 2004.

[28] S. Warshall. A theorem on boolean matrices. *Journal of the ACM (JACM)*, 9(1):11–12, 1962.

[29] R. R. Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the 46th ACM Symposium on Theory of Computing (STOC)*, pages 664–673, 2014.

[30] N. E. Young, R. E. Tarjan, and J. B. Orlin. Faster parametric shortest path and minimum-balance algorithms. *Networks*, 21(2):205–221, 1991.