Formal Methods in System Design
https://doi.org/10.1007/s10703-021-00373-5

Faster algorithms for quantitative verification in bounded treewidth graphs

Krishnendu Chatterjee[1] · Rasmus Ibsen-Jensen[2] · Andreas Pavlogiannis[3]

Received: 23 September 2015 / Accepted: 8 April 2021
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2021

Abstract

We consider the core algorithmic problems related to verification of systems with respect to three classical quantitative properties, namely, the mean-payoff, the ratio, and the minimum initial credit for energy property. The algorithmic problem given a graph and a quantitative property asks to compute the optimal value (the infimum value over all traces) from every node of the graph. We consider graphs with bounded treewidth—a class that contains the control flow graphs of most programs. Let n denote the number of nodes of a graph, m the number of edges (for bounded treewidth $m = O(n)$) and W the largest absolute value of the weights. Our main theoretical results are as follows. First, for the minimum initial credit problem we show that (1) for general graphs the problem can be solved in $O(n^2 \cdot m)$ time and the associated decision problem in $O(n \cdot m)$ time, improving the previous known $O(n^3 \cdot m \cdot \log(n \cdot W))$ and $O(n^2 \cdot m)$ bounds, respectively; and (2) for bounded treewidth graphs we present an algorithm that requires $O(n \cdot \log n)$ time. Second, for bounded treewidth graphs we present an algorithm that approximates the mean-payoff value within a factor of $1 + \epsilon$ in time $O(n \cdot \log(n/\epsilon))$ as compared to the classical exact algorithms on general graphs that require quadratic time. Third, for the ratio property we present an algorithm that for bounded treewidth graphs works in time $O(n \cdot \log(|a \cdot b|)) = O(n \cdot \log(n \cdot W))$, when the output is $\frac{a}{b}$, as compared to the previously best known algorithm on general graphs with running time $O(n^2 \cdot \log(n \cdot W))$. We have implemented some of our algorithms and show that they present a significant speedup on standard benchmarks.

The bulk of this work was done while R. Ibsen-Jensen and A. Pavlogiannis were at IST Austria.

✉ Andreas Pavlogiannis
pavlogiannis@cs.au.dk

Krishnendu Chatterjee
krishnendu.chatterjee@ist.ac.at

Rasmus Ibsen-Jensen
R.Ibsen-Jensen@liverpool.ac.uk

[1] IST Austria, Am Campus 1, 3400 Klosterneuburg, Austria

[2] University of Liverpool, Liverpool L69 3BX, UK

[3] Aarhus University, Aabogade 34, 8200 Aarhus, Denmark

Published online: 29 April 2021

Keywords Quantitative verification · Minimum mean cycle · Minimum ratio cycle · Constant treewidth graphs · Initial credit for energy

1 Introduction

Boolean vs quantitative verification The traditional view of verification has been *qualitative (Boolean)* that classifies traces of a system as "correct" vs "incorrect". In the recent years, motivated by applications to analyze resource-constrained systems (such as embedded systems), there has been a huge interest to study *quantitative* properties of systems. A quantitative property assigns to each trace of a system a real-number that quantifies how good or bad the trace is, instead of classifying it as correct vs incorrect. For example, a Boolean property may require that every request is eventually granted, whereas a quantitative property for each trace can measure the average waiting time between requests and corresponding grants.
Variety of results Given the importance of quantitative verification, the traditional qualitative view of verification has been extended in several ways, such as, quantitative languages and quantitative automata for specification languages [2,5,16,23,28,29,31,50]; quantitative logics for specification languages [1,9,11,32,38]; quantitative synthesis for robust reactive systems [4]; a framework for quantitative abstraction refinement [14]; and model measuring (that extends model checking) [36], to name a few. The core algorithmic question for many of the above studies is a graph algorithmic problem that requires to analyze a graph wrt a quantitative property.

Important quantitative properties The three quantitative properties that have been studied for their relevance in analysis of reactive systems are as follows. First, the *mean-payoff* property [27,37] consists of a weight function that assigns to every transition an integer-valued weight and assigns to each trace the long-run average of the weights of the transitions of the trace. Second, the *ratio* property [27] consists of two weight functions (one of which is a positive weight function) and assigns to each trace the ratio of the two mean-payoff properties (the denominator is wrt the positive function). The *minimum initial credit for energy* property [10] consists of a weight function (like in the mean-payoff property) and assigns to each trace the minimum number to be added such that the partial sum of the weights for every prefix of the trace is non-negative. For example, the mean-payoff property is used for average waiting time, worst-case execution time analysis [14,16,21]; the ratio property is used in robustness analysis of systems [4]; and the minimum initial credit for energy for measuring resource consumption [10].

Algorithmic problems Given a graph and a quantitative property, the value of a node is the infimum value of all traces that start at that node. The algorithmic problem (namely, the *value* problem) for analysis of quantitative properties consists of a graph and a quantitative property, and asks to compute either the exact value or an approximation of the value for every node in the graph. The algorithmic problems are at the heart of many applications, such as automata emptiness, model measuring, quantitative abstraction refinement, etc.

Treewidth of graphs A very well-known concept in graph theory is the notion of *treewidth* of a graph, which is a measure of how similar a graph is to a tree (a graph has treewidth 1 precisely if it is a tree) [45]. The treewidth of a graph is defined based on a *tree decomposition* of the graph [34], see Sect. 2 for a formal definition. Beyond the mathematical elegance of the treewidth property for graphs, there are many classes of graphs which arise in practice and have bounded treewidth. The most important example is that the control flow graphs of

Table 1 Complexity of the existing and our solution for the minimum initial credit problem on weighted graphs of n nodes, m edges, and largest absolute weight W

	Bouyer et. al. [10]	Our result [Thm 3, Cor 1]	Our result [Thm 5] (bounded treewidth)
Time (decision)	$O(n^2 \cdot m)$	$\mathbf{O(n \cdot m)}$	$\mathbf{O(n \cdot \log n)}$
Time	$O(n^3 \cdot m \cdot \log(n \cdot W))$	$\mathbf{O(n^2 \cdot m)}$	$\mathbf{O(n \cdot \log n)}$
Space	$O(n)$	$\mathbf{O(n)}$	$\mathbf{O(n)}$

Table 2 Time complexity of existing and our solutions for the minimum mean-cycle value and ratio-cycle value problem in bounded treewidth weighted graphs with n nodes and largest absolute weight W, when the output is the (irreducible) fraction $\frac{a}{b} \neq 0$

Minimum mean-cycle value			Minimum ratio-cycle value		
Orlin & Ahuja [43]	Karp [37]	Our result [Thm 7] (ϵ -approximate)	Burns [13]	Lawler [40]	Our result [Cor 3]
$O(n^{1.5} \cdot \log(n \cdot W))$	$O(n^2)$	$\mathbf{O(n \cdot \log(n/))}$	$O(n^3)$	$O(n^2 \cdot \log(n \cdot W))$	$\mathbf{O(n \cdot \log(\|a \cdot b\|))}$

goto-free programs for many programming languages are of bounded treewidth [48], and it was also shown in [33] that typically all Java programs have bounded treewidth. For many other applications see the surveys [6,7]. The bounded treewidth property of graphs has also played an important role in logic and verification; for example, MSO (Monadic Second Order logic) queries can be solved in polynomial time [24] (also in log-space [30]) for bounded-treewidth graphs; parity games on graphs with bounded treewidth can be solved in polynomial time [42]; and there exist faster algorithms for probabilistic models (like Markov decision processes) [15]. Moreover, it has been shown that the bounded treewidth property is also useful for interprocedural analysis [21] and intraprocedural analysis of concurrent programs [18].

Previous results and our contributions In this work we consider general graphs and graphs with bounded treewidth, and the algorithmic problems to compute the exact value or an approximation of the value for every node wrt to quantitative properties given as the mean-payoff, the ratio, or the minimum initial credit for energy. We first present the relevant previous results, and then our contributions.

Previous results. We consider graphs with n nodes, m edges, and let W denote the largest absolute value of the weights. The running time of the algorithms is characterized by the number of arithmetic operations (i.e., each operation takes constant time); and the space is characterized by the maximum number of integers the algorithm stores. The classical algorithm for graphs with mean-payoff properties is the minimum mean-cycle problem of Karp [37], and the algorithm requires $O(n \cdot m)$ running time and $O(n^2)$ space. A different algorithm was proposed in [41] that requires $O(n \cdot m)$ running time and $O(n)$ space. Orlin and Ahuja [43] gave an algorithm running in time $O(\sqrt{n} \cdot m \cdot \log(n \cdot W))$. For some special cases there exist faster approximation algorithms [19]. There is a straightforward reduction of the ratio problem to the mean-payoff problem. For computing the exact minimum ratio, the fastest known strongly polynomial time algorithm is Burns' algorithm [13] running in time $O(n^2 \cdot m)$. Also, there is an algorithm by Lawler [40] that uses $O(n \cdot m \cdot \log(n \cdot W))$ time. Many pseudopolynomial algorithms are known for the problem, with polynomial dependency on the numbers appearing in the weight function, see [27]. For the minimum initial credit for energy problem, the decision problem (i.e., is the energy required for node v at most c?) can be solved in $O(n^2 \cdot m)$ time, leading to an $O(n^3 \cdot m \cdot \log(n \cdot W))$ time algorithm for the minimum initial credit for energy problem [10]. All the above algorithms are for general graphs (without the bounded-treewidth restriction).

Our contributions. Our main contributions are as follows.

1. *Finding the minimum initial credit in graphs* We present two results. First, we consider the exact computation for general graphs, and present (i) an $O(n \cdot m)$ time algorithm for the decision problem (improving the previous known $O(n^2 \cdot m)$ bound), and (ii) an $O(n^2 \cdot m)$ time algorithm to compute value of all nodes (improving the previous known $O(n^3 \cdot m \cdot \log(n \cdot W))$ bound). Finally, we consider the computation of the exact value for graphs with bounded treewidth and present an algorithm that requires $O(n \cdot \log n)$ time (improving the previous known $O(n^4 \cdot \log(n \cdot W))$ bound) (see Table 1).
2. *Finding the mean-payoff and ratio values in bounded-treewidth graphs* We present two results for bounded treewidth graphs. First, for the exact computation we present an algorithm that requires $O(n \cdot \log(|a \cdot b|))$ time and $O(n)$ space, where $\frac{a}{b} \neq 0$ is the (irreducible) ratio/mean-payoff of the output. If $\frac{a}{b} = 0$ then the algorithm uses $O(n)$ time. Note that $\log(|a \cdot b|) \leq 2 \log(n \cdot W)$. We also present a space-efficient version of the algorithm that requires only $O(\log n)$ space. Second, we present an algorithm for finding an ϵ-factor approximation that requires $O(n \cdot \log(n/\epsilon))$ time and $O(n)$ space, as compared to the $O(n^{1.5} \cdot \log(n \cdot W))$ time solution of Orlin & Ahuja, and the $O(n^2)$ time solution of Karp (see Table 2).
3. *Experimental results* We have implemented our algorithms for the minimum mean cycle and minimum initial credit problems and ran them on standard benchmarks (DaCapo suit [3] for the minimum mean cycle problem, and DIMACS challenges [25] for the minimum initial credit problem). For the minimum mean cycle problem, our results show that our algorithm has lower running time than all the classical polynomial-time algorithms. For the minimum initial credit problem, our algorithm provides a significant speedup over the existing method. Both improvements are demonstrated even on graphs of small/medium size. Note that our theoretical improvements (better asymptotic bounds) imply improvements for large graphs, and our improvements on medium size graphs indicate that our algorithms have practical applicability with small constants.

Technical contributions The key technical contributions of our work are as follows:

1. *Minimum initial credit problem.* We show that for general graphs, the decision problem can be solved with two applications of Bellman-Ford-type algorithms, and the value problem reduces to finding non-positive cycles in the graph, followed by one instance of the single-source shortest path problem. We then show how the invariants of the algorithm for the value problem on general graphs can be maintained by a particular graph traversal of the tree-decomposition for bounded-treewidth graphs.
2. *Mean-payoff and ratio values in bounded-treewidth graphs.* Given a graph with bounded treewidth, let c^* be the smallest weight of a simple cycle. First, we present a linear-time algorithm that computes c^* exactly (if $c^* \geq 0$) or approximate within a polynomial factor (if $c^* < 0$). Then, we show that if the minimum ratio value ν^* is the irreducible fraction $\frac{a}{b}$, then ν^* can be computed by evaluating $O(\log(|a \cdot b|))$ inequalities of the form $\nu^* \geq \nu$. Each such inequality is evaluated by computing the smallest weight of a simple cycle in a modified graph. Finally, for ϵ-approximating the value ν^*, we show that $O(\log(n/\epsilon))$ such inequalities suffice.

A preliminary version of this work has appeared in [17].

2 Preliminaries

Weighted graphs We consider *finite weighted directed graphs* $G = (V, E, \mathsf{wt}, \mathsf{wt}')$ where V is the set of n *nodes*, $E \subseteq V \times V$ is the edge relation of m *edges*, $\mathsf{wt} : E \to \mathbb{Z}$ is a *weight function* that assigns an integer weight $\mathsf{wt}(e)$ to each edge $e \in E$, and $\mathsf{wt}' : E \to \mathbb{N}^+$ is a weight function that assigns strictly positive integer weights. For technical simplicity, we assume that there exists at least one outgoing edge from every node. In case where the function wt' is irrelevant, we will consider weighted graphs $G = (V, E, \mathsf{wt})$, i.e., without the function wt'.

Finite and infinite paths A *finite path* $P = (u_1, \dots, u_j)$, is a sequence of nodes $u_i \in V$ such that for all $1 \leq i < j$ we have $(u_i, u_{i+1}) \in E$. The *length* of P is $|P| = j - 1$. We often write $P : u_1 \rightsquigarrow u_j$ that P is a path from u_1 to u_j. A single-node path has length 0. The path P is *simple* if there is no node repeated in P, and it is a *cycle* if $j > 1$ and $u_1 = u_j$. The path P is a *simple cycle* if P is a cycle and the sequence $(u_2, \dots u_j)$ is a simple path. The functions wt and wt' naturally extend to paths, so that the weight of a path P with $|P| > 0$ wrt the weight functions wt and wt' is $\mathsf{wt}(P) = \sum_{1 \leq i < j} \mathsf{wt}(u_i, u_{i+1})$ and $\mathsf{wt}'(P) = \sum_{1 \leq i < j} \mathsf{wt}'(u_i, u_{i+1})$. The *value* of P is defined to be $\overline{\mathsf{wt}}(P) = \frac{\mathsf{wt}(P)}{\mathsf{wt}'(P)}$. For the case where $|P| = 0$, we define $\mathsf{wt}(P) = 0$, and $\overline{\mathsf{wt}}(P)$ is undefined. An *infinite path* $\mathcal{P} = (u_1, u_2, \dots)$ of G is an infinite sequence of nodes such that every finite prefix P of $\mathcal{P}$ is a finite path of G. The functions wt and wt' assign to $\mathcal{P}$ a value in $\mathbb{Z} \cup \{-\infty, \infty\}$: we have $\mathsf{wt}(\mathcal{P}) = \limsup_{j \to \infty} \sum_{i=1}^{j} \mathsf{wt}(u_i, u_{i+1})$ and $\mathsf{wt}'(\mathcal{P}) = \infty$. For a (possibly infinite) path P, we use the notation $u \in P$ to denote that a node u appears in P, and $e \in P$ to denote that an edge e appears in P. Given a set $B \subseteq V$, we denote by $P \cap B$ the set of nodes of B that appear in P. Given a finite path $P_1 = (u_1, \dots, u_j)$ and a possibly infinite path $P_2 = (u_j, \dots)$, we denote by $P_1 \circ P_2$ the path resulting from the concatenation of P_1 and P_2.

Distances and witness paths For nodes $u, v \in V$, we denote by $d(u, v) = \inf_{P : u \rightsquigarrow v} \mathsf{wt}(P)$ the *distance* from u to v. A finite path $P : u \rightsquigarrow v$ is a *witness* of the distance $d(u, v)$ if $\mathsf{wt}(P) = d(u, v)$. An infinite path $\mathcal{P}$ is a witness of the distance $d(u, v)$ if the following conditions hold:

1. $d(u, v) = \mathsf{wt}(\mathcal{P}) = -\infty$, and
2. $\mathcal{P}$ starts from u, and v is reachable from every node of $\mathcal{P}$.

Note that $d(u, v) = -\infty$ is only witnessed by infinite paths, whereas $d(u, v) = \infty$ is not witnessed by any path.

Tree decompositions A *tree-decomposition* [6,45] $\mathrm{Tree}(G) = (V_T, E_T)$ of G is a tree such that the following conditions hold:

1. $V_T = \{B_0, \dots, B_{n'-1} : \forall i \; B_i \subseteq V\}$ and $\bigcup_{B_i \in V_T} B_i = V$ (every node is covered).
2. For all $(u, v) \in E$ there exists $B_i \in V_T$ such that $u, v \in B_i$ (every edge is covered).
3. For all i, j, k such that there is a set B_k that appears in the simple path $B_i \rightsquigarrow B_j$ in $\mathrm{Tree}(G)$, we have $B_i \cap B_j \subseteq B_k$ (every node appears in a contiguous subtree of $\mathrm{Tree}(G)$).

The sets B_i which are nodes in V_T are called *bags*. Conventionally, we call B_0 the root of $\mathrm{Tree}(G)$, and denote by $\mathsf{Lv}(B_i)$ the level of B_i in $\mathrm{Tree}(G)$, with $\mathsf{Lv}(B_0) = 0$. We say that $\mathrm{Tree}(G)$ is *balanced* if the maximum level is $\max_{B_i} \mathsf{Lv}(B_i) = O(\log n')$, and it is *binary* if every bag has at most two children bags. A bag B is called the *root bag* of a node u if B is the smallest-level bag that contains u, and we often use B_u to refer to the root bag of u. The *width* of a tree-decomposition $\mathrm{Tree}(G)$ is the size of the largest bag minus 1. The treewidth of G is the smallest width among the widths of all possible tree decompositions of G. Figure 1

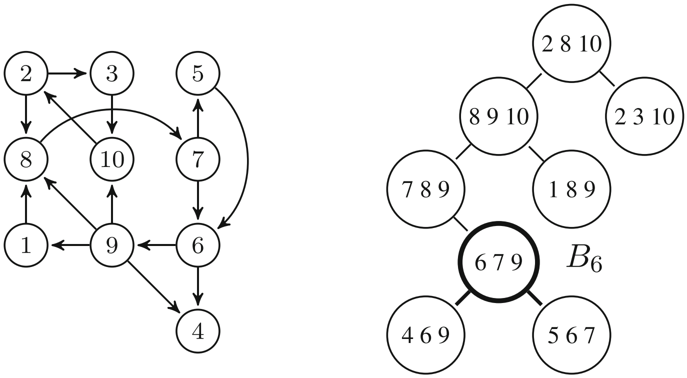

Fig. 1 A graph G with treewidth 2 (left) and a corresponding tree-decomposition $T = \text{Tree}(G)$ of 8 bags and width 2 (right). The distinguished bag B_6 is the root bag of node 6

illustrates the above definitions on an example. The following lemma gives a fundamental structural property of tree-decompositions.

Lemma 1 [46, (Proposition 2.6] *Consider a graph $G = (V, E)$, a binary tree-decomposition $T = \text{Tree}(G)$ and a bag B of T. Denote by $(\mathcal{C}_i)_{1\leq i\leq 3}$ the components of T created by removing B from T, and let V_i be the set of nodes that appear in bags of component $\mathcal{C}_i$. For every $i \neq j$, nodes $u \in V_i$, $v \in V_j$ and $P : u \rightsquigarrow v$, we have that $P \cap B \neq \emptyset$ (i.e., all paths between u and v go through some node in B).*

The following theorem gives the complexity of constructing balanced binary tree-decompositions of bounded-treewidth graphs.

Theorem 1 *For every graph G with n nodes and bounded treewidth $t = O(1)$, a balanced binary tree-decomposition* $\text{Tree}(G)$ *of $O(n)$ bags and*

1. *width $3 \cdot t + 2$ can be constructed in $O(n)$ time and space* [8],
2. *width $4 \cdot t + 3$ can be constructed in deterministic logspace (and hence polynomial time)* [30].

In the sequel we consider only balanced and binary tree-decompositions of bounded width and $n' = O(n)$ bags (and hence of height $O(\log n)$). Additionally, we assume that every bag is the root bag of at most one node. Obtaining this last property is straightforward, simply by replacing each bag B which is the root of $k > 1$ nodes $x_1, \dots x_k$ with a chain of bags $B_1, \dots, B_k = B$, where each B_i is the parent of B_{i+1}, and $B_{i+1} = B_i \cup \{x_{i+1}\}$. Note that this keeps the tree binary and increases its height by at most a constant factor, hence the resulting tree is also balanced.

Throughout the paper, we follow the convention that the maximum and minimum of the empty set is $-\infty$ and ∞ respectively, i.e., $\max(\emptyset) = -\infty$ and $\min(\emptyset) = \infty$. Time complexity is measured in number of arithmetic and logical operations, and space complexity is measured in number of machine words. Given a graph G, we denote by $\mathcal{T}(G)$ and $\mathcal{S}(G)$ the time and space required for constructing a balanced, binary tree-decomposition $\text{Tree}(G)$. We are interested in the following problems.

The minimum initial credit problem [10]. Given a weighted directed graph $G = (V, E, \mathsf{wt})$, the minimum initial credit value problem asks to determine for each node u the smallest *energy value* $\mathsf{E}(u) \in \mathbb{N} \cup \{\infty\}$ with the following property: there exists an infinite path $\mathcal{P} = (u_1, u_2 \dots)$ with $u = u_1$, such that for every finite prefix P of $\mathcal{P}$ we have $\mathsf{E}(u) + \mathsf{wt}(P) \geq 0$. Conventionally, we let $\mathsf{E}(u) = \infty$ if no finite value exists. The associated decision problem asks given a node u and an initial credit $c \in \mathbb{N}$ whether $\mathsf{E}(u) \leq c$.

The minimum mean cycle problem [37]. Given a weighted directed graph $G = (V, E, \mathsf{wt})$, the minimum mean cycle problem asks to determine for each node u the *mean value* $\mu^*(u) = \min_{C \in \mathcal{C}_u} \frac{\mathsf{wt}(C)}{|C|}$, where $\mathcal{C}_u$ is the set of simple cycles reachable from u in G. A cycle C with $\frac{\mathsf{wt}(C)}{|C|} = \mu^*(u)$ is called a minimum mean cycle of u. For $0 < \epsilon < 1$, we say that a value μ is an ϵ-approximation of the mean value $\mu^*(u)$ if $|\mu - \mu^*(u)| \leq \epsilon \cdot |\mu^*(u)|$.

The minimum ratio cycle problem [35]. Given a weighted directed graph $G = (V, E, \mathsf{wt}, \mathsf{wt}')$, the minimum ratio cycle problem asks to determine for each node u the *ratio value* $\nu^*(u) = \min_{C \in \mathcal{C}_u} \overline{\mathsf{wt}}(C)$, where $\overline{\mathsf{wt}}(C) = \frac{\mathsf{wt}(C)}{\mathsf{wt}'(C)}$ and $\mathcal{C}_u$ is the set of simple cycles reachable from u in G. A cycle C with $\overline{\mathsf{wt}}(C) = \nu_u^*$ is called a minimum ratio cycle of u. The minimum mean cycle problem follows as a special case of the minimum ratio cycle problem for $\mathsf{wt}'(e) = 1$ for each edge $e \in E$.

3 Minimum cycle

In the current section we deal with a related graph problem, namely the detection of a minimum-weight simple cycle of a graph. In Sect. 5 we use solutions to the minimum cycle problem to obtain the minimum ratio and minimum mean values of a graph.

The minimum cycle problem Given a weighted graph $G = (V, E, \mathsf{wt})$, the minimum cycle problem asks to determine the weight c^* of a minimum-weight simple cycle in G, i.e., $c^* = \min_{C \in \mathcal{C}} \mathsf{wt}(C)$, where $\mathcal{C}$ is the set of simple cycles in G.

We describe the algorithm MinCycle that operates on a tree-decomposition Tree(G) of an input graph G, and has the following properties.

1. If G has no negative cycles, then MinCycle returns the weight c^* of a minimum-weight cycle in G.
2. If G has negative cycles, then MinCycle returns a value that is at most a polynomial (in n) factor smaller than c^*.

U-*shaped paths* Following the work of [21], we define the important notion of U-shaped paths in a tree-decomposition Tree(G). Given a bag B and nodes $u, v \in B$, we say that a path $P : u \rightsquigarrow v$ is U-*shaped* in B, if one of the following conditions hold:

1. Either $|P| > 1$ and for all intermediate nodes $w \in P$, we have that B is an ancestor of B_w,
2. or $|P| \leq 1$ and B is B_u or B_v (i.e., B is the root bag of u or v).

Informally, given a bag B, a U-shaped path in B is a path that traverses intermediate nodes that exist only in the subtree of Tree(G) rooted in B. The following remark follows from the definition of tree-decompositions, and states that every simple cycle C can be seen as a U-shaped path P from the smallest-level node of C to itself. Consequently, we can determine the value c^* by only considering U-shaped paths in Tree(G).

Remark 1 Let $C = (u_1, \dots, u_k)$ be a simple cycle in G, and $u_j = \arg\min_{u_i \in C} \mathsf{Lv}(u_i)$. Then $P = (u_j, u_{j+1}, \dots u_k, u_1, \dots, u_j)$ is a U-shaped path in B_{u_j}, and $\mathsf{wt}(P) = \mathsf{wt}(C)$.

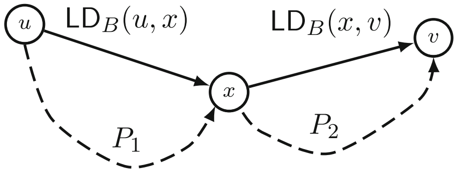

Fig. 2 Path shortening in line 10 of MinCycle. When B_x is examined, $\mathsf{LD}_{B_x}(u, v)$ is updated with the weight of the U-shaped path $P = P_1 \circ P_2$. The paths P_1 and P_2 are U-shaped paths in the children bags B_1 and B_2 of B_x, and we have $\mathsf{LD}_{B_i}(u, x) = \mathsf{wt}(P_i)$

Informal description of MinCycle Based on U-shaped paths, the work of [20] presented a method for computing algebraic path properties on tree-decompositions with bounded width, where the weights of the edges come from a general semiring. Note that integer-valued weights are a special case of the tropical semiring. Our algorithm MinCycle is similar to the algorithm Preprocess from [20]. It consists of a depth-first traversal of $\mathrm{Tree}(G)$, and for each examined bag B computes a *local distance* map $\mathsf{LD}_B : B \times B \to \mathbb{Z} \cup \{\infty\}$ such that for each $u, v \in B$, we have (i) $\mathsf{LD}_B(u, v) = \mathsf{wt}(P)$ for some path $P : u \rightsquigarrow v$, and (ii) $\mathsf{LD}_B(u, v) \leq \min_P \mathsf{wt}(P)$, where P are taken to be simple $u \rightsquigarrow v$ paths (or simple cycles) that are U-shaped in B. That is, the root bags of the intermediate nodes of P are descendants of B in $\mathrm{Tree}(G)$. This is achieved by traversing $\mathrm{Tree}(G)$ in post-order, and for each root bag B_x of a node x, we update every $\mathsf{LD}_{B_x}(u, v)$ with $\mathsf{LD}_{B_x}(u, x) + \mathsf{LD}_{B_x}(x, v)$, when the latter value is smaller than the former (i.e., we do path-shortening from node u to node v, by considering paths that go through x). See Fig. 2 for an illustration.

In the end, MinCycle returns $\min_x \mathsf{LD}_{B_x}(x, x)$, i.e., the weight of the smallest-weight U-shaped (not necessarily simple) cycle it has discovered. Algorithm 1 gives MinCycle in pseudocode. For brevity, in line 5 we consider that if $\{u, v\} \notin E$ or $\{u, v\} \not\subseteq B_i$ for some child B_i of B, then $\mathsf{LD}_{B_i}(u, v) = \infty$.

Algorithm 1: MinCycle

Input: A weighted graph $G = (V, E, \mathsf{wt})$ and a balanced binary tree-decomposition $\mathrm{Tree}(G)$
Output: A value c

```
1  Assign c ← ∞
2  Apply a post-order traversal on Tree(G), and examine each bag B with children B_1, B_2
3  begin
4      foreach u, v ∈ B do
5          Assign LD_B(u, v) ← min(LD_{B_1}(u, v), LD_{B_2}(u, v), wt(u, v))
6      end
7      Discard LD_{B_1}, LD_{B_2}
8      if B is the root bag of a node x then
9          foreach u, v ∈ B do
10             Assign LD'_B(u, v) ← min(LD_B(u, v), LD_B(u, x) + LD_B(x, v))
11         end
12         Assign LD_B ← LD'_B
13         Assign c ← min(c, LD_B(x, x))
14
15 end
16 return c
```

In essence, MinCycle performs repeated summarizations of paths in G.

Remark 2 Although most existing works follow this classic framework, they use multiple traversals of the tree-decomposition. The returned value of these methods can depend exponentially on n (rather than linearly, which is achieved by MinCycle).

We refer to Appendix Section A for the analysis MinCycle and proofs. The following states the correctness and complexity of MinCycle.

Theorem 2 *Let $G = (V, E, \mathsf{wt})$ be a weighted graph of n nodes with bounded treewidth, and a balanced, binary tree-decomposition* $\mathrm{Tree}(G)$ *of G be given. Let c^*, be the smallest weight of a simple cycle in G. Algorithm MinCycle uses $O(n)$ time and $O(\log n)$ additional space, and returns a value c such that:*

1. *If G has no negative cycles, then $c = c^*$.*
2. *If G has a negative cycle, then*
   (a) $c \le c^*$.
   (b) $|c| = |c^*| \cdot n^{O(1)}$.

4 The minimum initial credit problem

In the current section we present algorithms for solving the minimum initial credit problem on weighted graphs $G = (V, E, \mathsf{wt})$. We first deal with arbitrary graphs, and provide (i) an $O(n \cdot m)$ algorithm for the decision problem, and (ii) an $O(n^2 \cdot m)$ for the value problem, improving the previously best upper bounds. Afterwards we adapt our approach on graphs of bounded treewidth to obtain an $O(n \cdot \log n)$ algorithm for the value problem.

Non-positive minimum initial credit For technical convenience we focus on a variant of the minimum initial credit problem, where energies are non-positive, and the goal is to keep partial sums of path prefixes non-positive. Formally, given a weighted graph $G = (V, E, \mathsf{wt})$, the non-positive minimum initial credit value problem asks to determine for each node $u \in V$ the largest energy value $\mathsf{E}(u) \in \mathbb{Z}_{\le 0} \cup \{-\infty\}$ with the following property: there exists an infinite path $\mathcal{P} = (u_1, u_2 \dots)$ with $u = u_1$, such that for every finite prefix P of $\mathcal{P}$ we have $\mathsf{E}(u) + \mathsf{wt}(P) \le 0$. Conventionally, we let $\mathsf{E}(u) = -\infty$ if no finite such value exists. The associated decision problem asks given a node u and an initial credit $c \in \mathbb{Z}_{\le 0}$ whether $\mathsf{E}(u) \ge c$. Hence, here minimality is wrt the absolute value of the energy. A solution to the standard minimum initial credit problem can be obtained by inverting the sign of each edge weight and solving the non-positive minimum initial credit problem in the resulting graph.

We start with some definitions and lemmas that will give the intuition for the algorithms to follow. First, we define the minimum (wrt the absolute value) initial credit of a pair of nodes u, v, which is the energy to reach v from u (i.e., the energy is wrt a finite path).

Finite minimum initial credit For nodes $u, v \in V$, we denote by $\mathsf{E}_v(u) \in \mathbb{Z}_{\le 0} \cup \{-\infty\}$ the largest value with the following property: there exists a path $P : u \rightsquigarrow v$ such that for every prefix P' of P we have $\mathsf{E}_v(u) + \mathsf{wt}(P') \le 0$. Note that for every pair of nodes $u, v \in V$, we have $\mathsf{E}(u) \ge \mathsf{E}_v(u) + \mathsf{E}(v)$. Conventionally, we let $\mathsf{E}_v(u) = -\infty$ if no such value exists (i.e., there is no path $u \rightsquigarrow v$).

Remark 3 For any $u \in V$, let $P : u \rightsquigarrow v$ be a witness path for $\mathsf{E}_v(u) > -\infty$. Then

$$\mathsf{E}_v(u) + \mathsf{wt}(P) \le 0 \implies \mathsf{E}_v(u) \le -\mathsf{wt}(P) \le -d(u, v)$$

i.e., the energy to reach v from u is upper bounded by minus the distance from u to v.

Highest-energy nodes Given a (possibly infinite) path P with $\mathsf{wt}(P) < \infty$, we say that a node $x \in P$ is a *highest-energy node* of P if there exists a prefix P_1 of P ending in x (called a *highest-energy prefix*) such that for any prefix P_2 of P we have $\mathsf{wt}(P_1) \geq \mathsf{wt}(P_2)$. Note that since the weights are integers, for every pair of paths P_1', P_2', it is either $|\mathsf{wt}(P_1') - \mathsf{wt}(P_2')| = 0$ or $|\mathsf{wt}(P_1') - \mathsf{wt}(P_2')| \geq 1$. Therefore the set $\{\mathsf{wt}(P_i)\}_i$ of weights of prefixes of P has a maximum, and thus a highest-energy node always exists when $\mathsf{wt}(P) < \infty$.

Lemma 2 *The following properties hold.*

1. *If x is a highest-energy node in a path $P : u \rightsquigarrow v$, then $\mathsf{E}_v(x) = 0$.*
2. *If x is a highest-energy node in an infinite path $\mathcal{P}$, then $\mathsf{E}(x) = 0$.*

Proof 1. Indeed, let $P_1 : u \rightsquigarrow x$ be a prefix of P, and $P_2 : x \rightsquigarrow w$ any subpath of P such that $P_1 \circ P_2$ form a prefix of P. Then P_1 is a highest-energy prefix of P, and we have

$$\mathsf{wt}(P_1) \geq \mathsf{wt}(P_1 \circ P_2) = \mathsf{wt}(P_1) + \mathsf{wt}(P_2) \implies \mathsf{wt}(P_2) \leq 0$$

Since the last inequality holds for any path $P_2 : x \rightsquigarrow w$, we obtain that $\mathsf{E}_v(x) = 0$.

2. The proof is almost identical to the above case.

□

The following lemma states that the energy $\mathsf{E}(u)$ of a node u is the maximum energy $\mathsf{E}_v(u)$ to reach a 0-energy node v.

Lemma 3 *For every $u \in V$, we have $\mathsf{E}(u) = \max_{v:\mathsf{E}(v)=0} \mathsf{E}_v(u)$.*

Proof The direction $\mathsf{E}(u) \geq \max_{v:\mathsf{E}(v)=0} \mathsf{E}_v(u)$ is straightforward. For the other direction, consider that $\mathsf{E}(u) > -\infty$ (trivially, $-\infty \leq \max_{v:\mathsf{E}(v)=0} \mathsf{E}_v(u)$) and let $\mathcal{P}$ be a witness path for $\mathsf{E}(u)$. Since $\mathsf{E}(u) > -\infty$, we have $\mathsf{wt}(\mathcal{P}) < \infty$, and $\mathcal{P}$ has some highest-energy node x, thus by Lemma 2 we have $\mathsf{E}(x) = 0$. Since x is on the witness $\mathcal{P}$ of $\mathsf{E}(u)$, we have $\mathsf{E}(u) \leq \mathsf{E}_x(u) \leq \max_{v:\mathsf{E}(v)=0} \mathsf{E}_v(u)$. The result follows.

4.1 The decision problem for general graphs

Here we address the decision problem, namely, given some node $u \in V$ and an initial credit $c \in \mathbb{Z}_{\leq 0}$, determine whether $\mathsf{E}(u) \geq c$. The following lemma states that if $\mathsf{E}(u) \geq c$, then a non-positive cycle can be reached from u with initial credit c, by paths of length less than n.

Lemma 4 *For every $u \in V$ and $c \in \mathbb{Z}_{\leq 0}$, we have that $\mathsf{E}(u) \geq c$ iff there exists a simple cycle C such that (i) $\mathsf{wt}(C) \leq 0$ and (ii) for every $v \in C$ we have that $\mathsf{E}_v(u) \geq c$, which is witnessed by a path $P_v : u \rightsquigarrow v$ with $|P_v| < n$.*

Proof For the one direction, if $\mathsf{wt}(C) \leq 0$ we have $\mathsf{wt}(C^\omega) < \infty$, thus C contains a 0-energy node w. By Lemma 3, $\mathsf{E}(u) = \max_{v:\mathsf{E}(v)=0} \mathsf{E}_v(u) \geq \mathsf{E}_w(u) \geq c$. For the other direction, let $\mathcal{P}$ be a witness path for $\mathsf{E}(u)$, and we can assume w.l.o.g. that $\mathcal{P}$ does not contain positive cycles. Then for every prefix $P_w : u \rightsquigarrow v$ of $\mathcal{P}$ we have $\mathsf{E}(u) + \mathsf{wt}(P_w) \leq 0$, thus for every node v we have $\mathsf{E}_v(u) \geq \mathsf{E}(u) \geq c$, and the n-th such prefix contains a non-positive cycle C. The result follows. □

Algorithm DecisionEnergy Lemma 4 suggests a way to decide whether $\mathsf{E}(u) \geq c$. First, we start with energy c from u, and perform a sequence of $n - 1$ relaxation steps, similar to the Bellman-Ford algorithm, to discover the set V_u^c of nodes that can be reached from u with initial credit c by a path of length at most $n-1$. In essence, the algorithm computes, for increasing values of k, the shortest path P_v of length k from u to each other node v, with the additional property (as compared to Bellman-Ford) that the weight of every prefix of P_v is at most c. Afterwards, we perform a Bellman-Ford computation on the subgraph $G \upharpoonright V_u^c$ induced by the set V_u^c. By Lemma 4, we have that $\mathsf{E}(u) \geq c$ iff $G \upharpoonright V_u^c$ contains a non-positive cycle. Algorithm 2 (DecisionEnergy) gives a formal description. The *for* loop in lines 6-13 is similar to the procedure ROUND from the algorithm of [10].

Detecting non-positive cycles It is known that the Bellman-Ford algorithm can detect negative cycles. To detect non-positive cycles in a graph G with n nodes and weight function wt, we execute Bellman-Ford on G with a slightly modified weight function wt' for which $\mathsf{wt}'(e) = \mathsf{wt}(e) - \frac{1}{n}$. Then for any simple cycle C in G we have $\mathsf{wt}(C) \leq 0$ iff $\mathsf{wt}'(C) < 0$. Indeed,

$$\mathsf{wt}'(C) < 0 \iff \sum_{e \in C} \mathsf{wt}(e) - \sum_{e \in C} \frac{1}{n} < 0 \iff \mathsf{wt}(C) < \frac{|C|}{n} \iff \mathsf{wt}(C) \leq 0$$

since $|C| \leq n$ and $\mathsf{wt}(C) \in \mathbb{Z}$.

Algorithm 2: DecisionEnergy

Input: A weighted graph $G = (V, E, \mathsf{wt})$, a node $u \in V$, an initial energy $c \in \mathbb{Z}_{\leq 0}$
Output: True iff $\mathsf{E}(u) \geq c$

// Initialization
1 **foreach** $v \in V$ **do**
2   Assign $D(v) \leftarrow \infty$
3 **end**
4 Assign $D(u) \leftarrow c$
5 Assign $V_u^c \leftarrow \{u\}$
// $n-1$ relaxation steps to discover V_u^c
6 **for** $i \leftarrow 1$ **to** $n-1$ **do**
7   **foreach** $(v, w) \in E$ **do**
8   **if** $D(w) \geq D(v) + \mathsf{wt}(v, w)$ *and* $D(v) + \mathsf{wt}(v, w) \leq 0$ **then**
9   Assign $D(w) \leftarrow D(v) + \mathsf{wt}(v, w)$
10   Assign $V_u^c \leftarrow V_u^c \cup \{w\}$
11
12   **end**
13 **end**
14 Execute Bellman-Ford on $G \upharpoonright V_u^c$
15 **return** True iff a non-positive cycle is discovered

The correctness of DecisionEnergy follows directly from Lemma 4. The time complexity is $O(n \cdot m)$ time spent in the *for* loop of lines 6–13, plus $O(n \cdot m)$ time for the Bellman-Ford. We thus obtain the following theorem.

Theorem 3 *Let $G = (V, E, \mathsf{wt})$ be a weighted graph of n nodes and m edges. Let $u \in V$ be an initial node, and $c \in \mathbb{Z}_{\leq 0}$ be an initial credit. The decision problem of whether $\mathsf{E}(u) \geq c$ can be solved in $O(n \cdot m)$ time and $O(n)$ space.*

4.2 The value problem for general graphs

We now turn our attention to the value version of the minimum initial credit problem, where the task is to determine $\mathsf{E}(u)$ for every node u. The following lemma establishes that if all energies to reach some node v are negative, then the energy to reach v from every node u is minus the distance from u to v.

Lemma 5 *If for all $w \in V \setminus \{v\}$ we have $\mathsf{E}_v(w) < 0$, then for each $u \in V \setminus \{v\}$ we have $\mathsf{E}_v(u) = -d(u, v)$.*

Proof Let $P : u \rightsquigarrow v$ be a witness path to the distance, i.e., $\mathsf{wt}(P) = d(u, v) < \infty$ (if $d(u, v) = \infty$ the statement is trivially true). Since every highest-energy node x of P has $\mathsf{E}_v(x) = 0$ (Lemma 2), we have that $x = v$. Hence, P is a highest-energy prefix of itself, and for each prefix P' of P we have $-\mathsf{wt}(P) + \mathsf{wt}(P') \leq 0$ and thus by the definition of finite minimum initial credit, we have $\mathsf{E}_v(u) \geq -\mathsf{wt}(P) = -d(u, v)$. By Remark 3, it is $\mathsf{E}_v(u) \leq -d(u, v)$. The result follows. □

An $O(n^2 \cdot m)$ time solution to the value problem Lemma 5 together with Theorem 3 lead to an $O(n^2 \cdot m)$ method for solving the minimum initial credit value problem. First, we compute the set $X = \{v \in V : \mathsf{E}(v) = 0\}$ in $O(n^2 \cdot m)$ time, by testing whether $\mathsf{E}(u) \geq 0$ for each node u. Afterwards, we contract the set X to a new node z, and by Lemma 3 for every remaining node u we have $\mathsf{E}(u) = \max_{v \in X} \mathsf{E}_v(u) = \mathsf{E}_z(u)$. Since $u \notin X$, the energy of u is strictly negative, and thus $\mathsf{E}_z(u) < 0$. Finally, by Lemma 5, we have $\mathsf{E}_z(u) = -d(u, z)$. Hence it suffices to compute the distance of each node u to z, which can be obtained in $O(n \cdot m)$ time.

In the remainder of this subsection we provide a refined solution of $O(k \cdot n \cdot m)$ time, where $k = |X| + 1$ is the number of 0-energy nodes (plus one). Hence this solution is faster in graphs where $k = o(n)$. This is achieved by algorithm ZeroEnergyNodes for computing the set X faster.

Determining the 0-energy nodes The first step for solving the minimum initial credit problem is determining the set X of all 0-energy nodes of G. To achieve this, we construct the graph $G_2 = (V_2, E_2, \mathsf{wt}_2)$ with a fresh node $z \notin V$ as follows:

1. The node set is $V_2 = V \cup \{z\}$,
2. The edge set is $E_2 = E \cup (\{z\} \times V)$,
3. The weight function $\mathsf{wt}_2 : E_2 \to \mathbb{Z}$ is

$$\mathsf{wt}_2(u, v) = \begin{cases} 0 & \text{if } u = z \\ \mathsf{wt}(u, v) & \text{otherwise} \end{cases}$$

Remark 4 Since for every outgoing edge (z, x) of z we have $\mathsf{wt}_2(z, x) = 0$, if z is a highest-energy node in a path of G_2, so is x. Hence every non-positive cycle in G_2 has a highest-energy node other than z.

Note that for every $u \in V$, the energy $\mathsf{E}(u)$ is the same in G and G_2.

Algorithm ZeroEnergyNodes Algorithm 3 describes ZeroEnergyNodes for obtaining the set of all 0-energy nodes in G_2. Informally, the algorithm performs a sequence of modifications on a graph $\mathcal{G}$, initially identical to G_2. In each step, the algorithm executes a Bellman-Ford computation on the current graph $\mathcal{G}$ with z as the source node, as long as a non-positive cycle C is discovered. For every such C, it determines a highest-energy node w of C, inserts w to a set of discovered nodes $\mathcal{X}$, and modifies $\mathcal{G}$ by replacing every incoming edge (x, w) with

Algorithm 3: ZeroEnergyNodes

Input: A weighted graph $G_2 = (V_2, E_2, \mathsf{wt}_2)$
Output: The set $\{v \in V_2 \setminus \{z\} : \mathsf{E}(v) = 0\}$

1 Initialize sets $\mathcal{V} \leftarrow V_2$, $\mathcal{E} \leftarrow E_2$ and map $wt \leftarrow \mathsf{wt}_2$
2 Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, wt)$
3 Initialize set $\mathcal{X} \leftarrow \emptyset$
4 **while** *True* **do**
5   Execute Bellman-Ford from source node z in $\mathcal{G}$
6   **if** *exists non-positive cycle* C **then**
7   Determine a highest-energy node $w \neq z$ in C
8   Assign $\mathcal{X} \leftarrow \mathcal{X} \cup \{w\}$
9   **foreach** *edge* $(x, w) \in \mathcal{E}$ **do**
10   **if** $(x, z) \notin \mathcal{E}$ **then**
11   Assign $\mathcal{E} \leftarrow \mathcal{E} \cup \{(x, z)\}$
12   Assign $wt(x, z) \leftarrow \mathsf{wt}_2(x, w)$
13   **else**
14   Assign $wt(x, z) \leftarrow \min(\mathsf{wt}_2(x, w), wt(x, z))$
15   **end**
16   **end**
17   Assign $\mathcal{V} \leftarrow \mathcal{V} \setminus \{w\}$
18   **else**
19   **return** $\mathcal{X}$
20   **end**
21 **end**

an edge (x, z) of the same weight, and then removing w. Finally, the algorithm returns the set $\mathcal{X}$. See Fig. 3 for an illustration.

As 0-energy nodes are discovered, ZeroEnergyNodes performs a sequence of modifications to the graph $\mathcal{G}$. We denote by $\mathcal{G}^k$ the graph $\mathcal{G}$ after the k-th node has been added to $\mathcal{X}$ (and $\mathcal{G}^0 = G_2$). We also use the superscript-k in our graph notation to make it specific to $\mathcal{G}^k$ (e.g. $d^k(u, z)$ and $\mathsf{E}^k_z(u)$ denote respectively the distance from u to z, and the energy to reach z from u in $\mathcal{G}^k$). The following two lemmas establish the correctness of ZeroEnergyNodes.

Lemma 6 *For every $w \in \mathcal{X}$ we have $\mathsf{E}(w) = 0$.*

Proof The proof is by induction on the size of $\mathcal{X}$. It is trivially true when $|\mathcal{X}| = 0$. For the inductive step, let w be the $k+1$-st node added to $\mathcal{X}$. By line 7, w is a highest-energy node in a non-positive cycle C of $\mathcal{G}^k$. We split into two cases.

1. If $z \notin C$, then C is also a cycle of G, hence w is a highest-energy node in the infinite path $\mathcal{P} = C^\omega$ of G, and $\mathsf{E}(w) = 0$.
2. If $z \in C$, let x be the node before z in C. By the modifications of lines 11 and 14, it is $wt^k(x, z) = \mathsf{wt}_2(x, w')$, where w' is a node that has been added to $\mathcal{X}$ when the algorithm ran on $\mathcal{G}^i$ for some $i < k$. It follows that w is a highest-energy node in a path $P : z \rightsquigarrow w'$ in G_2, and thus a highest-energy node in a suffix $P' : w \rightsquigarrow w'$ of P, where P' is a path in G. Hence by Lemma 2 we have that $\mathsf{E}_{w'}(w) = 0$. By the induction hypothesis, w' is a 0-energy node, i.e., $\mathsf{E}(w') = 0$, thus by Lemma 3 we have $\mathsf{E}(w) \geq \mathsf{E}_{w'}(w) = 0$.

The result follows. □

Lemma 7 *For every $w \in V$ such that $\mathsf{E}(w) = 0$ we have $w \in \mathcal{X}$.*

Proof Consider any $w \in V$ such that $\mathsf{E}(w) = 0$. For some $i \in \mathbb{N}$, we say that $\mathcal{G}^i$ "is aware of w" if either $\mathcal{G}^i$ has a non-positive cycle $C : w \rightsquigarrow w$, or $w \in \mathcal{X}$ when $|\mathcal{X}| = i$. Note that when ZeroEnergyNodes terminates there are no non-positive cycles in $\mathcal{G}^{|\mathcal{X}|}$. Hence, it suffices to argue that there exists a $k \in \mathbb{N}$ such that for each $i \geq k$, $\mathcal{G}^i$ is aware of w. We first argue that there exists a k for which $\mathcal{G}^k$ is aware of w.

Let $\mathcal{P}$ be a witness for $\mathsf{E}(w) = 0$, hence $\mathcal{P}$ traverses a non-positive cycle C_1 in G, thus C_1 exists in $\mathcal{G}^0$. Then there exists a smallest $j \in \mathbb{N}$ such that some node w' of $\mathcal{P}$ is identified as a highest-energy node in a non-positive cycle C_2 of $\mathcal{G}^j$ (possibly $C_1 = C_2$) and is inserted to $\mathcal{X}$. If $w = w'$, we have that $\mathcal{G}^j$ is aware of w. Otherwise, since $\mathsf{E}(w) = 0$ and w' is a node in the witness $\mathcal{P}$, we have $\mathsf{E}_{w'}(w) = 0$. By the choice of w', the path $\mathcal{P}$ exists in $\mathcal{G}^j$, therefore $\mathsf{E}^j_{w'}(w) = \mathsf{E}_{w'}(w) = 0$, and by Remark 3, we have $d^j(w, w') \leq 0$. It is straightforward that after the modifications in lines 11 and 14, we have that $d^{j+1}(w, z) \leq d^j(w, w') \leq 0$, and since $wt^j(z, w) = \mathsf{wt}_2(z, w) = 0$, we have a non-positive cycle $C : w \rightsquigarrow w$ in $\mathcal{G}^{j+1}$ through z. Hence either $\mathcal{G}^j$ or $\mathcal{G}^{j+1}$ is aware of w, thus there exists a $k \in \mathbb{N}$ for which $\mathcal{G}^k$ is aware of w.

Finally, observe that the distance $d^i(w, z)$ does not increase in any $\mathcal{G}^i$ for $i \geq k$ until w is inserted to $\mathcal{X}$, hence for each $i \geq k$, the graph $\mathcal{G}^i$ is aware of w. The desired result follows. □

Lemmas 6 and 7 establish that $\mathcal{X} = X$, i.e. the set $\mathcal{X}$ returned by the algorithm is the set X of zero-energy nodes of G.

Determining the negative-energy nodes Having computed the set X of all the 0-energy nodes of G, the second step for solving the minimum initial credit value problem is to determine the energy of every other node $u \in V \setminus X$. Recall the graph $\mathcal{G}^{|X|} = (\mathcal{V}^{|X|}, \mathcal{E}^{|X|}, wt^{|X|})$ after the end of ZeroEnergyNodes.

Lemma 8 *For every $u \in V \setminus X$ we have $\mathsf{E}(u) = -d^{|X|}(u, z)$.*

Proof Consider any node $u \in V \setminus X = \mathcal{V}^{|X|} \setminus \{z\}$. By Lemma 5, in the graph G we have $\mathsf{E}(u) = \max_{v:\mathsf{E}(v)=0} \mathsf{E}_v(u)$, and by the correctness of ZeroEnergyNodes from Lemma 6 and Lemma 7 we have $X = \{v : \mathsf{E}(v) = 0\}$, thus $\mathsf{E}(u) = \max_{v \in X} \mathsf{E}_v(u)$. It is straightforward to verify that at the end of ZeroEnergyNodes, we have $\max_{v \in X} \mathsf{E}_v(u) = \mathsf{E}^{|X|}_z(u)$, i.e., the maximum energy to reach the set X in G is the energy to reach z in $\mathcal{G}^{|X|}$. For all $v \in \mathcal{V}^{|X|} \setminus \{z\}$ it is $\mathsf{E}^{|X|}_z(v) < 0$, otherwise we would have $\mathsf{E}(v) = 0$ and thus $v \in X$ and $v \notin \mathcal{V}^{|X|}$. Then by Lemma 5, $\mathsf{E}^{|X|}_z(u) = -d^{|X|}(u, z)$. We conclude that $\mathsf{E}(u) = -d^{|X|}(u, z)$. □

Hence, to compute the energy $\mathsf{E}(u)$ of every node $u \in V \setminus X$, it suffices to compute its distance to z in $\mathcal{G}^{|X|}$. This is straightforward by reversing the edges of $\mathcal{G}^{|X|}$ and performing a Bellman-Ford computation with z as the source node. Figure 3 illustrates the algorithms on a small example. We obtain the following theorem.

Theorem 4 *Let $G = (V, E, \mathsf{wt})$ be a weighted graph of n nodes and m edges, and $k = |\{v \in V : E(v) = 0\}| + 1$. The minimum initial credit value problem for G can be solved in $O(k \cdot n \cdot m)$ time and $O(n)$ space.*

Proof Lemmas 6, 7 and 8 establish the correctness, so it remains to argue about the complexity. The *while* block of line 4 is executed at most once for each 0-energy node, hence at most k times. Inside the block, the execution of Bellman-Ford in line 5 requires $O(n \cdot m)$ time and $O(m)$ space. Since the Bellman-Ford algorithm uses backpointers to remember predecessors of nodes in distances, a highest-energy node w of a non-positive cycle C in line 7 can be

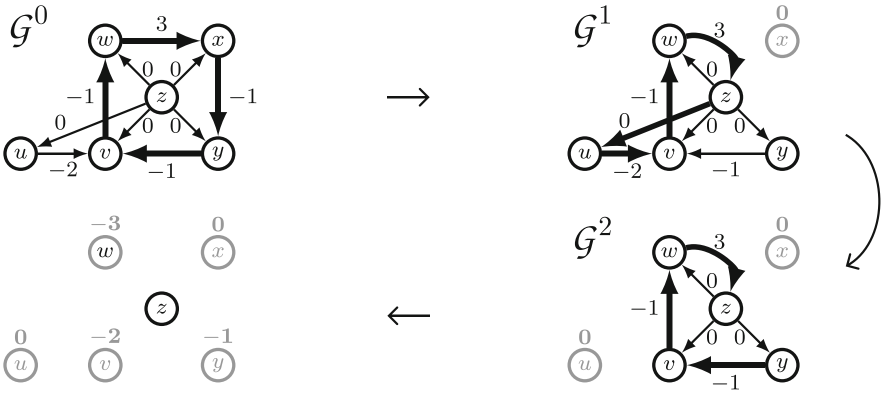

Fig. 3 Solving the value problem using operations on the graph $\mathcal{G}$. Initially we examine $\mathcal{G}^0$, and a non-positive cycle is found (boldface edges) with highest-energy node x. Thus $\mathsf{E}(x) = 0$, and we proceed with $\mathcal{G}^1$, to discover $\mathsf{E}(u) = 0$. In $\mathcal{G}^2$ all cycles are positive, and the energy of each remaining node is minus its distance to z

determined in $O(n)$. Finally, the *for* loop of line 9 will consider each edge (x, w) at most once, hence it requires $O(m)$ for all iterations of the *while* loop. Thus ZeroEnergyNodes uses $O(k \cdot n \cdot m)$ time and $O(n)$ space in total. The last execution of Bellman-Ford to determine the energy of negative-energy nodes does not affect the complexity. The result follows. □

Corollary 1 *Let $G = (V, E, \mathsf{wt})$ be a weighted graph of n nodes and m edges. The minimum initial credit value problem for G can be solved in $O(n^2 \cdot m)$ time and $O(n)$ space.*

4.3 The value problem for bounded-treewidth graphs

We now turn our attention to the minimum initial credit value problem for bounded-treewidth graphs $G = (V, E, \mathsf{wt})$. Note that in such graphs $m = O(n)$, thus Theorem 4 gives an $O(n^3)$ time solution as compared to the existing $O(n^4 \cdot \log(n \cdot W))$ time solution. This section shows that we can do significantly better, namely reduce the time complexity to $O(n \cdot \log n)$. This is mainly achieved by algorithm ZeroEnergyNodesTW for computing the set X of 0-energy nodes fast in bounded-treewidth graphs.

Extended **+** *and* **min** *operators* Recall the graph $G_2 = (V_2, E_2, \mathsf{wt}_2)$ from the last section. Given Tree(G), a balanced and binary tree-decomposition Tree(G_2) of G_2 with width increased by 1 can be easily constructed by (i) inserting z to every bag of Tree(G), and (ii) adding a new root bag that contains only z. Let $\mathcal{I} = \mathbb{Z} \times V \times \mathbb{Z}$. For a map $f : V_2 \times V_2 \to \mathbb{Z}$, define the map $g_f : V_2 \times V_2 \to \mathcal{I}$ as

$$g_f(u, v) = \begin{cases} (f(u, v), u, 0) & \text{if } f(u, v) < 0 \text{ or } v = z \\ (f(u, v), v, f(u, v)) & \text{otherwise} \end{cases}$$

and for triplets of elements $\alpha_1 = (a_1, b_1, c_1)$, $\alpha_2 = (a_2, b_2, c_2) \in \mathcal{I}$, define the operations

1. $\mathbf{min}(\alpha_1, \alpha_2) = \alpha_i$ with $i = \arg\min_{j \in \{1,2\}} a_j$
2. $\alpha_1 \mathbf{+} \alpha_2 = (a_1 + a_2, b, c)$, where $c = \max(c_1, a_1 + c_2)$ and $b = b_1$ if $c = c_1$ else $b = b_2$.

In words, if f is a weight function, then $g_f(u, v)$ selects the weight of the edge (u, v), and its highest-energy node (i.e., u if $f(u, v) < 0$, and v otherwise, except when $v = z$), together

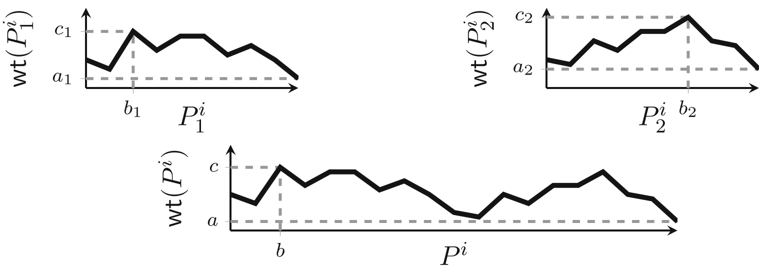

Fig. 4 Illustration of the $\alpha_1 \texttt{+} \alpha_2$ operation, corresponding to concatenating paths P_1 and P_2. The path P_j^i denotes the i-th prefix of P_j. We have $P = P_1 \circ P_2$, and the corresponding triplet $\alpha = (a, b, c)$ denotes the weight a of P, its highest-energy node b, and the weight c of a highest-energy prefix

with the weight to reach that highest energy node from u. Recall that algorithm MinCycle from Sect. 3 traverses a tree-decomposition bottom-up, and for each encountered bag B stores a map LD_B such that $\mathsf{LD}_B(u, v)$ is upper bounded by the weight of the shortest U-shaped simple path $u \rightsquigarrow v$ (or simple cycle, if $u = v$). Our algorithm ZeroEnergyNodesTW for determining all 0-energy nodes is similar, only that now LD_B stores triplets (a, b, c) where a is the weight of a U-shaped path P, b is a highest-energy node of P, and c the weight of a highest-energy prefix of P. For two triplets $\alpha_1 = (a_1, b_1, c_1), \alpha_2 = (a_2, b_2, c_2) \in \mathcal{I}$ corresponding to U-shaped paths P_1 and P_2, $\mathbf{min}(\alpha_1, \alpha_2)$ selects the path with the smallest weight, and $\alpha_1 \texttt{+} \alpha_2$ determines the weight, a highest-energy node, and the weight of a highest-energy prefix of the path $P_1 \circ P_2$ (see Fig. 4).

Algorithm ZeroEnergyNodesTW The algorithm ZeroEnergyNodesTW for computing the set of 0-energy nodes in bounded-treewidth graphs follows the same principle as ZeroEnergyNodes for general graphs. It stores a map of edge weights $wt : E_2 \to \mathbb{Z} \cup \{\infty\}$, and initially $wt(u, v) = \mathsf{wt}_2(u, v)$ for each $(u, v) \in E_2$. The algorithm performs a bottom-up pass, and computes in each bag the local distance map $\mathsf{LD}_B : B \times B \to \mathcal{I}$ that captures U-shaped $u \rightsquigarrow v$ paths, together with their highest-energy nodes. When a non-positive cycle C is found in some bag B, the method KillCycle is called to modify the edges of a highest-energy node w of C and its incoming neighbors by updating the map wt. These updates generally affect the distances between the rest of the nodes in the graph, hence some local distance maps LD_B need to be corrected. However, each such edge modification only affects the local distance map of bags that appear in a path from a bag B' to some ancestor B'' of B'. Instead of restarting the computation as in ZeroEnergyNodes, the method Update is called to correct those local distance maps along the path $B' \rightsquigarrow B''$.

The following lemma establishes the correctness of ZeroEnergyNodesTW. Similarly as for Lemma 6 and Lemma 7 we denote by $\mathcal{G}^k$ the graph obtained by considering the edges (u, v) for which $wt(u, v) < \infty$ when $|X| = k$.

Lemma 9 *For every $v \in V \setminus \{z\}$ we have $v \in X$ iff $\mathsf{E}(v) = 0$.*

Proof We only need to argue that ZeroEnergyNodesTW correctly computes the non-positive cycles in every $\mathcal{G}^k$, as then the correctness follows from the correctness Lemma 6 and Lemma 7 of ZeroEnergyNodes. Since by Remark 1 every cycle is a U-shaped path in some bag, it suffices to argue that whenever ZeroEnergyNodesTW examines a bag B (either directly, or through Update), every U-shaped simple cycle in B has been considered by the algorithm. This is true if no calls to KillCycle are made (*if* block in line 19), as then ZeroEnergyNodesTW is the same as MinCycle, and hence it follows from Lemma 14.

Algorithm 4: ZeroEnergyNodesTW

Input: A weighted graph $G_2 = (V_2, E_2, \mathsf{wt}_2)$ and a binary tree-decomposition $\mathrm{Tree}(G_2)$
Output: The set $\{v \in V_2 \setminus \{z\} : \mathsf{E}(v) = 0\}$

// Initialization
Assign $X \leftarrow \emptyset$
foreach $u, v \in V_2$ **do**
  if $(u, v) \in E_2$ **then**
    Assign $wt(u, v) \leftarrow \mathsf{wt}_2(u, v)$
  else
    Assign $wt(u, v) \leftarrow \infty$
  end
end
// Computation
Apply a post-order traversal on $\mathrm{Tree}(G)$, and examine each bag B with children B_1, B_2
begin
  foreach $u, v \in B$ **do**
    Assign $\mathsf{LD}_B(u, v) \leftarrow \mathbf{min}(\mathsf{LD}_{B_1}(u, v), \mathsf{LD}_{B_2}(u, v), g_{wt}(u, v))$
  end
  if *B is the root bag of a node x* **then**
    foreach $u, v \in B$ **do**
      Assign $\mathsf{LD}'_B(u, v) \leftarrow \mathbf{min}(\mathsf{LD}_B(u, v), \mathsf{LD}_B(u, x)+\mathsf{LD}_B(x, v))$
    end
    Assign $\mathsf{LD}_B \leftarrow \mathsf{LD}'_B$
    if *$\exists u \in B$ with $LD_B(u, u) = (a, b, c)$ where $a \le 0$* **then**
      Assign $X \leftarrow X \cup \{b\}$
      Execute KillCycle on b and B
end
return X

Method 5: KillCycle

Input: A 0-energy node w and a bag B of $\mathrm{Tree}(G_2)$
Output: Updates the local distance function LD_B

1 **foreach** *edge* $(x, w) \in E_2$ **do**
2   Assign $wt(x, z) \leftarrow \min(\mathsf{wt}_2(x, w), wt(x, z))$
3   Assign $wt(x, w) \leftarrow \infty$
4   Assign $y \leftarrow \arg\max_{u \in \{x, w\}} \mathsf{Lv}(u)$
5   Let B' be the smallest-level ancestor of B_y examined by ZeroEnergyNodesTW so far
6   Execute Update on B_y and its ancestor B'
7 **end**
8 **return** LD_B

Now consider that KillCycle is called and B' is the smallest-level bag examined by ZeroEnergyNodesTW so far. Let w be the 0-energy node, x an incoming neighbor of w, and $y = \arg\max_{u \in \{x, w\}} \mathsf{Lv}(u)$ (as in line 4 of KillCycle). By the definition of U-shaped paths, the edge (x, w) appears only in paths that are U-shaped in bags along the path $B_y \rightsquigarrow B'$. Hence, after setting $wt(x, w) = \infty$ (line 3 of KillCycle), it suffices to update the local distance maps of these bags. Similarly, after setting $wt(x, z) \leftarrow \min(\mathsf{wt}_2(x, w), wt(x, z))$ (line 2 of KillCycle), since B_z is the root of $\mathrm{Tree}(G_2)$, it suffices to update the local distance maps in the bags along the path $B_x \rightsquigarrow B'$. Either $x = y$, or, by the properties of tree-decompositions,

Method 6: Update

Input: A bag B' and an ancestor B''
Output: The local distances LD_B along the path $B' \rightsquigarrow B''$

Traverse the path $B' \rightsquigarrow B''$ bottom-up, and examine each bag B with children B_1, B_2
begin
  foreach $u, v \in B$ **do**
    Assign $\mathsf{LD}_B(u, v) \leftarrow \mathbf{min}(\mathsf{LD}_{B_1}(u, v), \mathsf{LD}_{B_2}(u, v), g_{wt}(u, v))$
  end
  if *B is the root bag of a node x* **then**
    foreach $u, v \in B$ **do**
      Assign $\mathsf{LD}'_B(u, v) \leftarrow \mathbf{min}(\mathsf{LD}_B(u, v), \mathsf{LD}_B(u, x)+\mathsf{LD}_B(x, v))$
    end
    Assign $\mathsf{LD}_B \leftarrow \mathsf{LD}'_B$
    if *$\exists u \in B$ with $\mathsf{LD}_B(u, u) = (a, b, c)$ where $a \leq 0$* **then**
      Assign $X \leftarrow X \cup \{b\}$
      Execute KillCycle on b and B
end

B_x is an ancestor of B_y. Hence in either case $B_x \rightsquigarrow B'$ is a subpath of $B_y \rightsquigarrow B'$, and both edge modifications in lines 2 and 3 are handled correctly by calling Update on B_y and its ancestor B'. The result follows. □

Lemma 10 *Algorithm ZeroEnergyNodesTW runs in $O(n \cdot \log n)$ time and $O(n)$ space.*

Proof Let $h = O(\log n)$ be the height of $\mathrm{Tree}(G_2)$.

1. The method Update performs a constant number of operations to each bag in the path $B' \rightsquigarrow B''$ where B'' is ancestor of B', hence each call to Update requires $O(h)$ time.
2. The method KillCycle performs a constant number of operations locally and one call to Update for each incoming edge of w. Hence if w has k_w incoming edges, KillCycle requires $O(h \cdot k_w)$ time. Since KillCycle sets $wt(x, w) = \infty$ for all incoming edges of w, the node w will not appear in non-positive cycles thereafter.
3. The algorithm ZeroEnergyNodesTW is similar to MinCycle which runs in $O(n)$ time and space (Lemma 17). The difference is in the additional *if* block in line 19. Since KillCycle is called when a non-positive cycle is detected, it will be called at most once for each node $u \in V_2 \setminus \{z\}$ (from either ZeroEnergyNodesTW or Update). It follows that the total time of ZeroEnergyNodesTW is
$$O\left(n + \sum_u (h \cdot k_u)\right) = O(n + h \cdot |E_2|) = O(n \cdot \log n)$$
where k_u is the number of incoming edges of node u. Since KillCycle stores constant amount of information in each bag of $\mathrm{Tree}(G_2)$, the $O(n)$ space bound follows.

□

After the set X of 0-energy nodes has been computed, it remains to execute one instance of the single-source shortest path problem on the graph $\mathcal{G}^{|X|}$ (similarly as for our solution on general graphs). It is known that single-source distances in tree-decompositions of bounded treewidth can be computed in $O(n)$ time [21,22]. We thus obtain the following theorem.

Theorem 5 *Let $G = (V, E, \mathsf{wt})$ be a weighted graph of n nodes with bounded treewidth. The minimum initial credit value problem for G can be solved in $O(n \cdot \log n)$ time and $O(n)$ space.*

5 The minimum ratio and mean cycle problems

In the current section we present algorithms for solving the minimum ratio and mean cycle problems for weighted graphs $G = (V, E, \mathsf{wt}, \mathsf{wt}')$ of bounded treewidth.

Remark 5 If G is not strongly connected, we can compute its maximal strongly connected components (SCCs) in linear time [47], and use the algorithms of this section to compute the minimum cycle ratio ν_i^* in every component $\mathcal{G}_i$. Afterwards, we assign the ratio values $\nu^*(u)$ for all nodes u as follows. First, mark every SCC $\mathcal{G}_i$ with $M(\mathcal{G}_i) = \nu_i^*$. Then, for every bottom SCC $\mathcal{G}_i$, (i) for every u in $\mathcal{G}_i$ assign $\nu^*(u) = M(\mathcal{G}_i)$, (ii) for every neighbor SCC $\mathcal{G}_j$ of $\mathcal{G}_i$, mark $\mathcal{G}_j$ with $M(\mathcal{G}_j) = \min(M(\mathcal{G}_j), M(\mathcal{G}_i))$, (iii) remove $\mathcal{G}_i$ and repeat. Since these operations require linear time in total, they do not impact the time complexity. We also argue that we can focus on SCCs while using $O(\log n)$ space. Using [30], we can solve directed s-t connectivity in logspace for bounded-treewidth graphs. For any node v, let $\mathsf{SCC}(v)$ denote the strongly connected component of v. For each v we can find the value of the minimum ratio cycle when the graph is restricted to $\mathsf{SCC}(v)$, using any algorithm that can solve the problem if the graph is strongly connected, by simply ignoring all nodes w such that v cannot reach w or w cannot reach v. Then, for any node u, the value $\nu^*(u)$ is computed by solving the minimum ratio cycle problem restricted in $\mathsf{SCC}(v)$, for every node v reachable from u, and returning the minimum of all these values. Therefore, we consider graphs G that are strongly connected, and we will speak about the minimum ratio ν^* and mean μ^* values of G.

Lemma 11 *Let ν^* be the ratio value of G. Then $\nu^* \geq \nu$ iff for every cycle C of G we have $\mathsf{wt}_\nu(C) \geq 0$, where $\mathsf{wt}_\nu(e) = \mathsf{wt}(e) - \mathsf{wt}'(e) \cdot \nu$ for each edge $e \in E$.*

Proof Indeed, for any cycle C we have $\overline{\mathsf{wt}}(C) \geq \nu^* \geq \nu$. Then

$$\overline{\mathsf{wt}}(C) \geq \nu \iff \overline{\mathsf{wt}}(C) - \nu \geq 0 \iff \frac{\mathsf{wt}(C) - \nu \cdot \mathsf{wt}'(C)}{\mathsf{wt}'(C)} \geq 0$$
$$\iff \mathsf{wt}(C) - \nu \cdot \mathsf{wt}'(C) \geq 0 \iff \sum_{e \in C} (\mathsf{wt}(e) - \mathsf{wt}'(e) \cdot \nu) \geq 0 \iff \mathsf{wt}_\nu(C) \geq 0$$

with the equality holding iff $\overline{\mathsf{wt}}(C) = \nu$. □

Hence, given a tree-decomposition $\mathrm{Tree}(G)$, for any guess ν of the ratio value ν^*, we can evaluate whether $\nu^* \geq \nu$ by constructing the weight function $\mathsf{wt}_\nu = \mathsf{wt} - \nu$ and executing algorithm $\mathsf{MinCycle}$ on input $G_\nu = (V, E, \mathsf{wt}_\nu)$. By Item (a) of Theorem 2 and Lemma 11 we have that the returned value c of $\mathsf{MinCycle}$ is $c \geq 0$ iff $\mathsf{wt}_\nu(C) \geq 0$ for all cycles C, iff $\nu^* \geq \nu$ (and in fact $c = 0$ iff $\nu^* = \nu$).

Lemma 12 *Let $G = (V, E, \mathsf{wt}, \mathsf{wt}')$ be a weighted graph of n nodes with bounded treewidth and minimum ratio value ν^*. Let $\mathrm{Tree}(G)$ be a given balanced, binary tree-decomposition of G of bounded width. For any rational ν, the decision problem of whether $\nu^* \geq \nu$ (or $\nu^* = \nu$) can be solved in $O(n)$ time and $O(\log n)$ extra space.*

Proof By Lemma 11, we can test whether $\nu^* \geq \nu$ by testing whether $G_\nu = (V, E, \mathsf{wt}_\nu)$ has a negative cycle. By Theorem 2, a non-positive cycle in G_ν can be detected in $O(n)$ time and using $O(\log n)$ space. In particular, if the weight of the cycle found by MinCycle is 0, we have $\nu^* = \nu$, as mentioned above. □

5.1 Exact solution

We now describe the method for determining the value ν^* of G exactly. This is done by making various guesses ν such that $\nu^* \geq \nu$ and testing for negative cycles in the graph $G_\nu = (V, E, \mathsf{wt}_\nu)$. We first determine whether $\nu^* = 0$, using Lemma 12. In the remainder of this section we assume that $\nu^* \neq 0$.
Solution overview Consider that $\nu^* > 0$. First, we either find that $\nu^* \in (0, 1)$ (hence $\lfloor \nu^* \rfloor = 0$), or perform an *exponential search* of $O(\log \nu^*)$ iterations to determine $j \in \mathbb{N}^+$ such that $\nu^* \in [2^{j-1}, 2^j]$. In the latter case, we perform a binary search of $O(\log \nu^*)$ iterations in the interval $[2^{j-1}, 2^j]$ to determine $\lfloor \nu^* \rfloor$. Then, we can write $\nu^* = \lfloor \nu^* \rfloor + x$, where $x < 1$ is an irreducible fraction $\frac{a'}{b}$. It has been shown [44] that such x can be determined by evaluating $O(\log b)$ inequalities of the form $x \geq \nu$. The case for $\nu^* < 0$ is handled similarly. We refer to Appendix Section B for details and formal proofs.

Theorem 6 *Let $G = (V, E, \mathsf{wt}, \mathsf{wt}')$ be a weighted graph of n nodes with bounded treewidth, and $\lambda = \max_u |a_u \cdot b_u|$ such that $\nu^*(u)$ is the irreducible fraction $\frac{a_u}{b_u}$. Let $\mathcal{T}(G)$ and $\mathcal{S}(G)$ denote the required time and space for constructing a balanced binary tree-decomposition* Tree(G) *of G with bounded width. The minimum ratio cycle problem for G can be computed in*

1. *$O(\mathcal{T}(G) + n \cdot \log(\lambda))$ time and $O(\mathcal{S}(G) + n)$ space; and*
2. *$O(\mathcal{S}(G) + \log n)$ space.*

Using Theorem 1 we obtain from Theorem 6 the following corollary.

Corollary 2 *Let $G = (V, E, \mathsf{wt}, \mathsf{wt}')$ be a weighted graph of n nodes with bounded treewidth, and $\lambda = \max_u |a_u \cdot b_u|$ such that $\nu^*(u)$ is the irreducible fraction $\frac{a_u}{b_u}$. The minimum ratio value problem for G can be computed in*

1. *$O(n \cdot \log(\lambda))$ time and $O(n)$ space; and*
2. *$O(\log n)$ space.*

By setting $\mathsf{wt}'(e) = 1$ for each $e \in E$ in Corollary 2 we obtain the following corollary for the minimum mean cycle.

Corollary 3 *Let $G = (V, E, \mathsf{wt})$ be a weighted graph of n nodes with bounded treewidth, and $\lambda = \max_u |\mu^*(u)|$. The minimum mean value problem for G can be computed in*

1. *$O(n \cdot \log(\lambda))$ time and $O(n)$ space; and*
2. *$O(\log n)$ space.*

5.2 Approximating the minimum mean cycle

We now focus on the minimum mean cycle problem, and present algorithms for ϵ-approximating the mean value μ^* of G for any $0 < \epsilon < 1$ in $O(n \cdot \log(n/\epsilon))$ time, i.e., independent of μ^*.

Approximate solution in the absence of negative cycles We first consider graphs G that do not have negative cycles. Let C be a minimum mean value cycle, and C' a minimum weight simple cycle in G, and note that $\mu^* \in [0, \mathsf{wt}(C')]$. Additionally, we have

$$\mathsf{wt}(C') \leq \mathsf{wt}(C) \implies \mathsf{wt}(C') \leq \frac{n}{|C|} \cdot \mathsf{wt}(C) \implies \mathsf{wt}(C') \leq n \cdot \mu^*$$

Consider a binary search in the interval $[0, \mathsf{wt}(C')]$, which in step i approximates μ^* by the right endpoint μ_i of its current interval. The error is bounded by the length of the interval, hence $\mu_i - \mu^* \leq \mathsf{wt}(C') \cdot 2^{-i} \leq (n-1) \cdot \mu^* \cdot 2^{-i}$. To approximate within a factor ϵ we require

$$2^{-i} \cdot (n-1) \leq \epsilon \implies i \geq \log(n) + \log(1/\epsilon) \tag{1}$$

steps.

Remark 6 Note that for the minimum ratio value we have $\mathsf{wt}(C') \leq W' \cdot n \cdot \nu^*$, where $W' = \max_{e \in E} \mathsf{wt}'(e)$. For ϵ-approximating ν^* we would need $i \geq \log(n \cdot W'/\epsilon)$ steps.

Approximate solution in the presence of negative cycles We now turn our attention to ϵ-approximating μ^* in the presence of negative cycles in G. Note that uniformly increasing the weight of each edge so that no negative edges exist does not suffice, as the error can be of order $\epsilon \cdot |W^-|$ rather than $\epsilon \cdot \mu^*$, where W^- is the minimum edge weight.

Instead, let c be the value returned by $\mathsf{MinCycle}$ on input G. Item (a) of Theorem 2 guarantees that for the weight function $\mathsf{wt}_{-|c|}(e) = \mathsf{wt}(e) + |c|$, the graph $G_{-|c|} = (V, E, \mathsf{wt}_{-|c|})$ has no negative cycles (although it might still have negative edges). The following lemma states that μ^* can be ϵ-approximated by ϵ'-approximating the value μ'^* of $G_{-|c|}$, for some ϵ' polynomially (in n) smaller than ϵ.

Lemma 13 *Let μ^* and μ'^* be the value of G and $G_{-|c|}$ respectively, and ϵ some desired approximation factor of μ^*, with $0 < \epsilon < 1$. There exists an $\epsilon' = \epsilon/n^{O(1)}$ such that if μ' is an ϵ'-approximation of μ'^* in $G_{-|c|}$, then $\mu = \mu' - |c|$ is an ϵ-approximation of μ^* in G.*

Proof By construction, we have $\mu'^* = \mu^* + |c|$, where c defined above is the value returned by $\mathsf{MinCycle}$ on G. Let c^* be the weight of a minimum-weight simple cycle in G. By Theorem 2 Item (b), we have that $|c| = |c^*| \cdot n^{O(1)}$. Note that $|c^*| \leq (n-1) \cdot |\mu^*|$, hence $\mu'^* = \mu^* + |c^*| \cdot n^{O(1)} \leq |\mu^*| \cdot \alpha$ for $\alpha = n^{O(1)}$. Let $\epsilon' = \epsilon/\alpha$. By ϵ'-approximating μ'^* by μ' we have

$$\begin{aligned} |\mu' - \mu'^*| \leq \epsilon' \cdot |\mu'^*| &\implies |(\mu' - |c|) - (\mu'^* - |c|)| \leq \epsilon' \cdot |\mu'^*| \\ &\implies |\mu - \mu^*| \leq \epsilon' \cdot |\mu^*| \cdot \alpha \leq \epsilon \cdot |\mu^*| \end{aligned}$$

The desired result follows. □

Theorem 7 *Let $G = (V, E, \mathsf{wt})$ be a weighted graph of n nodes with bounded treewidth. For any $0 < \epsilon < 1$, the minimum mean value problem can be ϵ-approximated in $O(n \cdot \log(n/\epsilon))$ time and $O(n)$ space.*

Proof In view of Remark 5 the graph G is strongly connected and has a minimum mean value μ^*. First, we construct a balanced binary tree-decomposition $\mathrm{Tree}(G)$ of G in $O(n)$ time and space using Theorem 1. Let c be the value returned by $\mathsf{MinCycle}$ on the input graph G. If $c \geq 0$, by Lemma 16 we have $\mu^* \geq 0$, and by Eq. 1 μ^* can be ϵ-approximated in $O(\log(n/\epsilon))$ steps. If $c < 0$, we construct the graph $G_{-|c|} = (V, E, \mathsf{wt}_{-|c|})$. By Lemma 13, μ^* can be ϵ-approximated by ϵ' approximating the mean value μ'^* of $G_{-|c|}$, where $\epsilon' = \frac{\epsilon}{n^{O(1)}}$. By

Table 3 Asymptotic complexity of compared minimum mean cycle algorithms

	Madani [41]	Burns [13]	Lawler [40]	Dasdan-Gupta [26]	Hartmann-Orlin [35]	Karp [37]
Time	$O(n^2)$	$O(n^3)$	$O(n^2 \cdot \log(n \cdot W))$	$O(n^2)$	$O(n^2)$	$O(n^2)$
Space	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n^2)$

construction, $G_{-|c|}$ does not contain negative cycles, thus $\mu'^* \geq 0$, and by Eq. 1 μ'^* can be approximated in $O(\log(n/\epsilon')) = O(\log(n/\epsilon))$ steps. By Lemma 17, each step requires $O(n)$ time. The statement follows. □

6 Experimental results

In the current section we report on preliminary experimental evaluation of our algorithms, and compare them to existing methods. Our algorithm for the minimum mean cycle problem provides improvement for bounded-treewidth graphs, and has thus been evaluated on low-treewidth graphs obtained from the control-flow graphs of programs. For the minimum initial credit problem, we have implemented our algorithm for arbitrary graphs, thus the benchmarks used in this case are general graphs (i.e., arbitrary-treewidth graphs).

6.1 Minimum mean cycle

We have implemented our approximation algorithm for the minimum mean cycle problem, and we let the algorithm run for as many iterations until a minimum mean cycle was discovered, instead of terminating after $O(\log(n/\epsilon))$ iterations required by Theorem 7. We have tested its performance in running time and space against six other minimum mean cycle algorithms from Table 3 in control-flow graphs of programs. The algorithms of Burns and Lawler solve the more general ratio cycle problem, and have been adapted to the mean cycle problem as in [27].

Setup The algorithms were executed on control-flow graphs of methods of programs from the DaCapo benchmark suit [3], obtained using the Soot framework [49]. For each benchmark we focused on graphs of at least 500 nodes. This supplied a set of medium sized graphs (between 500 and 1300 nodes), in which integer weights were assigned uniformly at random in the range $\{-10^3, \ldots, 10^3\}$. Memory usage was measured with [12].

Results Figure 5 shows the average time and space performance of the examined algorithms (bars that exceeded the maximum value in the y-axis have been truncated). Our algorithm has much smaller running time than each other algorithm, in almost all cases. In terms of space, our algorithm outperforms three other algorithms (i.e., all except for the algorithms of Lawler, Burns, and Madani). Both ours and these three algorithms have linear space complexity, but ours also suffers some constant factor overhead from the tree-decomposition (i.e., the same node generally appears in multiple bags). Note that the strong performance of these three algorithms in space is followed by poor performance in running time.

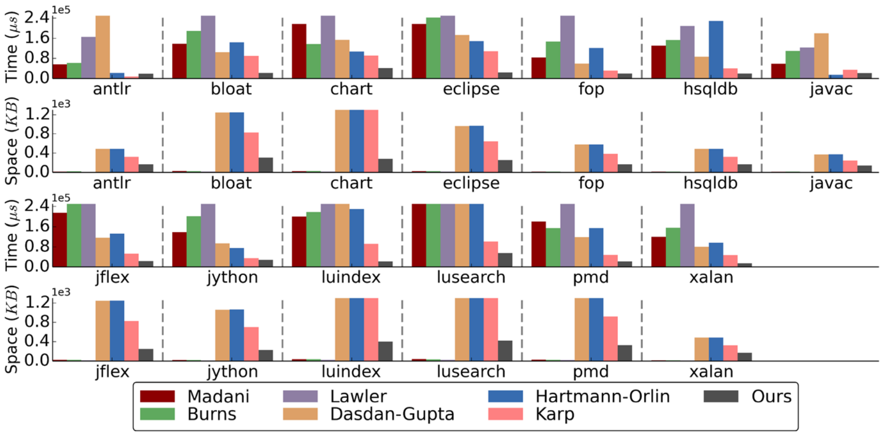

Fig. 5 Average performance of minimum mean cycle algorithms

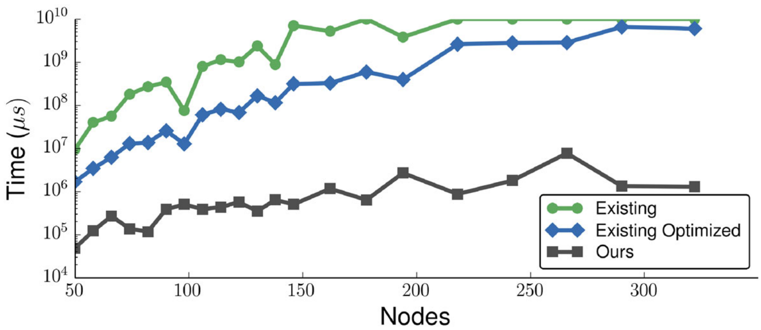

Fig. 6 Comparison of running times for the minimum initial credit problem

6.2 Minimum initial credit

We have implemented our algorithm for the minimum initial credit problem on general graphs and experimentally evaluated its performance on a subset of benchmark weighted graphs from the DIMACS implementation challenges [25]. Our algorithm was tested against the existing method of [10]. The direct implementation of the algorithm of [10] performed poorly, and for this we also implemented an optimized version (using techniques such as caching of intermediate results and early loop termination). Note that we compare algorithms for general graphs, without the low-treewidth restriction.

Setup For each input graph we first computed its minimum mean value μ^* using Karp's algorithm, and then subtracted μ^* from the weight of each edge to ensure that at least one non-positive cycle exists (thus the energies are finite).

Results Figure 6 depicts the running time of the algorithm of [10] (with and without optimizations) vs our algorithm. A timeout was forced at $10^{10}\mu s$. Our algorithm is orders of magnitude faster, and scales better than the existing method.

Acknowledgements The research was partly supported by Austrian Science Fund (FWF) Grant No P23499-N23, FWF NFN Grant No S11407-N23 (RiSE/SHiNE), ERC Start Grant (279307: Graph Games), and Microsoft faculty fellows award.

A Minimum cycle

Here we present in full detail the algorithm MinCycle and establish its properties as stated in Theorem 2. Recall that MinCycle operates on a tree-decomposition Tree(G) of an input graph G and behaves as follows.

1. If G has no negative cycles, then MinCycle returns the weight c^* of a minimum-weight cycle in G.
2. If G has negative cycles, then MinCycle returns a value that is at most a polynomial (in n) factor smaller than c^*.

Intuitively, MinCycle discovers (not necessarily simple) cycles in G and returns the minimum weight found. For reasons of efficiency, MinCycle does not keep track of which examined cycles are simple, hence if G has negative cycles, then MinCycle might return a value smaller than c^*, which corresponds to an examined non-simple cycle C that traverses some negative cycles multiple times. However, C is guaranteed to have polynomial (in n) length, and thus the weight of C is at most polynomially smaller than c^*.

The following lemma follows easily from [20, Lemma 2], and states that $\mathsf{LD}_B(u, v)$ is upper bounded by the smallest weight of a U-shaped simple $u \rightsquigarrow v$ path in B.

Lemma 14 ([20, Lemma 2]) *For every examined bag B and nodes $u, v \in B$, we have*

1. *$\mathsf{LD}_B(u, v) = \mathsf{wt}(P)$ for some path $P : u \rightsquigarrow v$ (and $\mathsf{LD}_B(u, v) = \infty$ if no such P exists),*
2. *$\mathsf{LD}_B(u, v) \leq \min_{P:u\rightsquigarrow v} \mathsf{wt}(P)$ where P ranges over U-shaped simple paths and simple cycles in B.*

At the end of the computation, the returned value c is the weight of a (generally non-simple) cycle C, captured as a U-shaped path on its smallest-level node. The cycle C can be recovered by tracing backwards the updates of line 10 performed by the algorithm, starting from the node x that performed the last update in line 13. Hence, if C traverses k distinct edges, we can write

$$c = \mathsf{wt}(C) = \sum_{i=1}^{k} k_i \cdot \mathsf{wt}(e_i) \tag{2}$$

where each e_i is a distinct edge, and k_i is the number of times it appears in C.

Lemma 15 *Let h be the height of* Tree(G). *For every k_i in Eq. 2, we have $k_i \leq 2^h$.*

Proof Note that the edge $e_i = (u_i, v_i)$ is first considered by MinCycle in the root bag B_i of node x_i, where $x_i = \arg\max_{y_i \in \{u_i, v_i\}} \mathsf{Lv}(y_i)$ (line 10). As MinCycle backtracks from B_i to the root of Tree(G), the edge e_i can be traversed at most twice as many times in each step (because of line 10, once for each term of the sum $\mathsf{LD}_B(u, x) + \mathsf{LD}_B(x, v)$). Hence, this doubling will occur at most h times, and thus $k_i \leq 2^h$. □

Lemma 16 *Let c be the value returned by MinCycle, h be the height of* Tree(G), *and $c^* = \min_C \mathsf{wt}(C)$ over all simple cycles C in G. The following assertions hold:*

1. *If G has no negative cycles, then $c = c^*$.*
2. *If G has a negative cycle, then*
   (a) *$c \leq c^*$.*
   (b) *$|c| = O\left(|c^*| \cdot n \cdot 2^h\right)$.*

Proof By Remark 1, we have that $c^* = \mathsf{wt}(P)$ for a U-shaped path $P : x \rightsquigarrow x$. By Lemma 14, after MinCycle examines B_x, it will be $c \leq \mathsf{LD}_{B_x}(x, x) \leq c^*$, with the equalities holding if there are no negative cycles in G (by the definition of c^*, as then $\mathsf{LD}_{B_x}(x, x)$ is witnessed by a simple cycle). By line 10, c can only decrease afterwards, and again by the definition of c^* this can only happen if there are negative cycles in G. This proves items 1 and 2a, and the remaining of the proof focuses on showing that $|c| = O\left(|c^*| \cdot n \cdot 2^h\right)$.

By rearranging the sum of Eq. 2, we can decompose the obtained cycle C into a set of k'^+ non-negative simple cycles C_i^+, and a set of k'^- negative simple cycles C_i^-, and each cycle C_i^+ and C_i^- appears with multiplicity k_i^+ and k_i^- respectively. Then we have

$$|c| = |\mathsf{wt}(C)| = \left|\sum_{i=1}^{k'^+} k_i^+ \cdot \mathsf{wt}(C_i^+) + \sum_{i=1}^{k'^-} k_i^- \cdot \mathsf{wt}(C_i^-)\right| \leq \left|\sum_{i=1}^{k'^-} k_i^- \cdot \mathsf{wt}(C_i^-)\right|$$

$$\leq \sum_{i=1}^{k^-} k_i^- \cdot |\mathsf{wt}(C_i^-)| \leq |c^*| \cdot \sum_{i=1}^{k'^-} k_i^- \leq |c^*| \cdot \sum_{i=1}^{k} k_i = O\left(|c^*| \cdot n \cdot 2^h\right) \tag{3}$$

The first inequality follows from $c < 0$, the third inequality holds by the definition of c^*, and the last inequality holds since the total number of (non-positive) simple cycle traversals of C cannot be more than the total number of the edge traversals. Finally, we have $\sum_{i=1}^{k} k_i = O\left(n \cdot 2^h\right)$, since $k = O(n)$, and by Lemma 15 we have $k_i \leq 2^h$. □

Next we discuss the time and space complexity of MinCycle.

Lemma 17 *Let h be the height of* $\mathrm{Tree}(G)$. *MinCycle accesses each bag of* $\mathrm{Tree}(G)$ *a constant number of times, and uses* $O(h)$ *additional space.*

Proof MinCycle accesses each bag a constant number of times, as it performs a post-order traversal on $\mathrm{Tree}(G)$ (line 2). Because it computes the local distances in a postorder manner, the number of local distance maps LD_B it remembers is bounded by the height h of $\mathrm{Tree}(G)$. Since $\mathrm{Tree}(G)$ has bounded width, LD_B requires a constant number of words for storing a constant number of nodes and weights in each B. Hence the total space usage is $O(h)$, and the result follows. □

The following theorem summarizes the results of this section.

Theorem 2 *Let* $G = (V, E, \mathsf{wt})$ *be a weighted graph of n nodes with bounded treewidth, and a balanced, binary tree-decomposition* $\mathrm{Tree}(G)$ *of G be given. Let* c^**, be the smallest weight of a simple cycle in G. Algorithm MinCycle uses* $O(n)$ *time and* $O(\log n)$ *additional space, and returns a value c such that:*

1. *If G has no negative cycles, then* $c = c^*$.
2. *If G has a negative cycle, then*
   (a) $c \leq c^*$.
   (b) $|c| = |c^*| \cdot n^{O(1)}$.

B The minimum ratio and mean cycle problems

Lemma 18 *Let* $\nu^* \neq 0$ *be the ratio value of G. The value* $\lfloor \nu^* \rfloor$ *can be obtained by evaluating* $O(\log |\nu^*|)$ *inequalities of the form* $\nu^* \geq \nu$.

Proof First determine whether $\nu^* > 0$, and assume w.l.o.g. that this is the case (the process is similar if $\nu^* < 0$). Perform an exponential search on the interval $(0, 2 \cdot \lfloor \nu^* \rfloor)$ by a sequence of evaluations of the inequality $\nu^* \geq \nu_i = 2^i$. After $\log \lfloor \nu^* \rfloor + 1$ steps we either have $\lfloor \nu^* \rfloor \in (0, 1)$, or have determined a $j > 0$ such that $\nu^* \in [\nu_{j-1}, \nu_j]$. Then, perform a binary search in the interval $[\nu_{j-1}, \nu_j]$, until the running interval $[\ell, r]$ has length at most 1. Since $\nu_j - \nu_{j-1} = \nu_{j-1} \leq \nu^*$, this will happen after at most $\log \lceil \nu^* \rceil$ steps. Then either $\lfloor \nu^* \rfloor = \lfloor \ell \rfloor$ or $\lfloor \nu^* \rfloor = \lfloor r \rfloor$, which can be determined by evaluating the inequality $\nu^* \geq \lfloor r \rfloor$. A similar process can be carried out when $\nu^* < 0$. □

Let $T_{\max} = \max_e \mathsf{wt}'(e)$ be the largest weight of an edge with respect to wt'. Since ν^* is a number with denominator at most $(n-1) \cdot T_{\max}$, it can be determined exactly by carrying the binary search of Lemma 18 until the length of the running interval becomes at most $\frac{1}{((n-1) \cdot T_{\max})^2}$ (thus containing a unique rational with denominator at most $(n-1) \cdot T_{\max}$). Then ν^* can be obtained by using continued fractions, e.g. as in [39]. We rely in the work of Papadimitriou [44] to obtain a tighter bound.

Lemma 19 *Let $\nu^* \neq 0$ be the ratio value of G, such that ν^* is the irreducible fraction $\frac{a}{b} \in (-1, 1)$. Then ν^* can be determined by evaluating $O(\log b)$ inequalities of the form $\nu^* \geq \nu$.*

Proof Consider that $\nu^* > 0$ (the proof is similar when $\nu^* < 0$). It is shown in [44] that a rational with denominator at most b can be determined by evaluating $O(\log b)$ inequalities of the form $\nu^* \geq \nu$. □

Theorem 6 *Let $G = (V, E, \mathsf{wt}, \mathsf{wt}')$ be a weighted graph of n nodes with bounded treewidth, and $\lambda = \max_u |a_u \cdot b_u|$ such that $\nu^*(u)$ is the irreducible fraction $\frac{a_u}{b_u}$. Let $\mathcal{T}(G)$ and $\mathcal{S}(G)$ denote the required time and space for constructing a balanced binary tree-decomposition* $\mathrm{Tree}(G)$ *of G with bounded width. The minimum ratio cycle problem for G can be computed in*

1. *$O(\mathcal{T}(G) + n \cdot \log(\lambda))$ time and $O(\mathcal{S}(G) + n)$ space; and*
2. *$O(\mathcal{S}(G) + \log n)$ space.*

Proof In view of Remark 5 the graph G is strongly connected and has a minimum ratio value ν^*. Let $\nu^* = \lfloor \nu^* \rfloor + \frac{a'}{b}$ with $|\frac{a'}{b}| < 1$. By Lemma 18, $\lfloor \nu^* \rfloor$ can be determined by evaluating $O(\log |\nu^*|) = O(\log |a|)$ inequalities of the form $\nu^* \geq \nu$, and by Lemma 19, $\frac{a'}{b}$ can be determined by evaluating $O(b)$ such inequalities. A balanced binary tree-decomposition $\mathrm{Tree}(G)$ can be constructed once in $\mathcal{T}(G)$ time and $\mathcal{S}(G)$ space, and stored in $O(n)$ space. $\mathrm{Tree}(G)$ is also a tree-decomposition of every G_ν required by Lemma 11. By Theorem 2 a negative cycle in G_ν can be detected in $O(n)$ time and using $O(\log n)$ space. This concludes Item 1. Item 2 is obtained by the same process, but with re-computing $\mathrm{Tree}(G)$ every time $\mathsf{MinCycle}$ traverses from a bag to a neighbor (thus not storing $\mathrm{Tree}(G)$ explicitly). □

References

1. Almagor S, Boker U, Kupferman O (2013) Formalizing and reasoning about quality. ICALP. LNCS. Springer, New York, pp 15–27
2. Alur R, D'Antoni L, Deshmukh JV, Raghothaman M, Yuan Y (2013) Regular functions and cost register automata. In: LICS, pp 13–22

3. Blackburn SM, Garner R, Hoffmann C, Khang AM, McKinley KS, Bentzur R, Diwan A, Feinberg D, Frampton D, Guyer SZ, Hirzel M, Hosking A, Jump M, Lee H, Moss JEB, Phansalkar A, Stefanović D, VanDrunen T, von Dincklage D, Wiedermann B (2006) The DaCapo benchmarks: Java benchmarking development and analysis. In: OOPSLA. ACM, pp 169–190
4. Bloem R, Chatterjee K, Greimel K, Henzinger TA, Hofferek G, Jobstmann B, Könighofer B, Könighofer R (2014) Synthesizing robust systems. Acta Inf 51(3–4):193–220
5. Blumensath A, Colcombet T, Kuperberg D, Parys P, Vanden Boom M (2014) Two-way cost automata and cost logics over infinite trees. In: CSL-LICS, pp 16:1–16:9
6. Bodlaender HL (1993) A tourist guide through treewidth. Acta Cybern
7. Bodlaender H (2005) Discovering treewidth. In: SOFSEM: theory and practice of computer science. LNCS. Springer
8. Bodlaender H, Hagerup T (1995) Parallel algorithms with optimal speedup for bounded treewidth. In: ICALP. LNCS. Springer
9. Bollig B, Gastin P, Monmege B, Zeitoun M (2014) Pebble weighted automata and weighted logics. ACM Trans Comput Log 15(2):15
10. Bouyer P, Fahrenberg U, Larsen KG, Markey N, Srba J (2008) Infinite runs in weighted timed automata with energy constraints. In: FORMATS. LNCS, vol 5215. Springer, pp 33–47
11. Bouyer P, Markey N, Matteplackel RM (2014) Averaging in LTL. CONCUR. LNCS. Springer, pp 266–280
12. Brosius D. Java agent for memory measurements. https://github.com/jbellis/jamm
13. Burns SM (1991) Performance analysis and optimization of asynchronous circuits. Technical report
14. Cerny P, Henzinger TA, Radhakrishna A (2013) Quantitative abstraction refinement. In: POPL. ACM, pp 115–128
15. Chatterjee K, Lacki J (2013) Faster algorithms for Markov decision processes with low treewidth. In: CAV. LNCS. Springer
16. Chatterjee K, Doyen L, Henzinger TA (2010) Quantitative languages. Trans Comput Log
17. Chatterjee K, Ibsen-Jensen R, Pavlogiannis A (2015) Faster algorithms for quantitative verification in bounded treewidth graphs. In: CAV, pp 140–157
18. Chatterjee K, Goharshady AK, Ibsen-Jensen R, Pavlogiannis A (2016) Algorithms for algebraic path properties in concurrent systems of constant treewidth components. In: POPL, pp 733–747
19. Chatterjee K, Henzinger M, Krinninger S, Loitzenbauer V, Raskin MA (2014) Approximating the minimum cycle mean. Theor Comput Sci
20. Chatterjee K, Ibsen-Jensen R, Pavlogiannis A, Goyal P (2015) Faster algorithms for algebraic path properties in recursive state machines with constant treewidth. In: POPL. ACM
21. Chatterjee K, Pavlogiannis A, Velner Y (2015) Quantitative interprocedural analysis. In: POPL, pp 539–551
22. Chaudhuri S, Zaroliagis CD (1995) Shortest paths in digraphs of small treewidth. Sequential algorithms, algorithmica, part I
23. Colcombet T, Löding C (2010) Regular cost functions over finite trees. In: LICS, pp 70–79
24. Courcelle B (1990) The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Inf Comput 85:12–75
25. DIMACS implementation challenges. http://dimacs.rutgers.edu/Challenges/
26. Dasdan A, Gupta R (1998) Faster maximum and minimum mean cycle algorithms for system-performance analysis. IEEE Trans Comput Aided Des Integr Circuits Syst 17:889–899
27. Dasdan A, Irani SS, Gupta RK (1998) An experimental study of minimum mean cycle algorithms. Technical report
28. Droste M, Kuich W, Vogler H (2009) Handbook of weighted automata. Springer, New York
29. Droste M, Meinecke I (2012) Weighted automata and weighted MSO logics for average and long-time behaviors. Inf Comput 220:44–59
30. Elberfeld M, Jakoby A, Tantau T (2010) Logspace versions of the theorems of Bodlaender and Courcelle. In: FOCS. IEEE Computer Society
31. Filiot E, Gentilini R, Raskin JF (2012) Quantitative languages defined by functional automata. In: Koutny M, Ulidowski I (eds) CONCUR 2012-concurrency theory. Springer, Berlin, pp 132–146
32. Forejt V, Kwiatkowska M, Norman G, Parker D, Qu H (2011) Quantitative multi-objective verification for probabilistic systems. In: Abdulla PA, Leino KRM (eds) Tools and algorithms for the construction and analysis of systems. Springer, Berlin, pp 112–127
33. Gustedt J, Mhle O, Telle J (2002) The treewidth of java programs. In: Algorithm engineering and experiments. LNCS. Springer
34. Halin R (1976) S-functions for graphs. J Geom 8:171–186
35. Hartmann M, Orlin JB (1993) Finding minimum cost to time ratio cycles with small integral transit times. Networks 23:567–574

36. Henzinger TA, Otop J (2013) From model checking to model measuring. In: CONCUR. LNCS. Springer, pp 273–287
37. Karp RM (1978) A characterization of the minimum cycle mean in a digraph. Discrete Math 23:309–311
38. Kreutzer S, Riveros C (2013) Quantitative monadic second-order logic. In: Proceedings of the 2013 28th annual ACM/IEEE symposium on logic in computer science. LICS ’13. IEEE Computer Society, New York, pp 113–122
39. Kwek S, Mehlhorn K (2003) Optimal search for rationals. Inf Process Lett 86(1):23–26
40. Lawler E (1976) Combinatorial optimization: networks and matroids. Saunders College Publishing, Philadelphia
41. Madani O (2002) Polynomial value iteration algorithms for deterministic MDPs. In: UAI. Morgan Kaufmann Publishers
42. Obdrzálek J (2003) Fast mu-calculus model checking when tree-width is bounded. In: CAV. LNCS. Springer
43. Orlin JB, Ahuja RK (1992) New scaling algorithms for the assignment and minimum mean cycle problems. Math Program 54:41–56
44. Papadimitriou CH (1979) Efficient search for rationals. In: IPL, pp 1–4
45. Robertson N, Seymour P (1984) Graph minors. III. Planar tree-width. J Combin Theory Ser B 36:49–64
46. Robertson N, Seymour P (1986) Graph minors. II. Algorithmic aspects of tree-width. J Algorithms 7:309–322
47. Tarjan R (1972) Depth-first search and linear graph algorithms. SIAM J Comput 1:146–160
48. Thorup M (1998) All structured programs have small tree width and good register allocation. Inf Comput 142:159–181
49. Vallée-Rai R, Co P, Gagnon E, Hendren L, Lam P, Sundaresan V (1999) Soot—a java bytecode optimization framework. In: CASCON ’99. IBM Press
50. Velner Y (2012) The complexity of mean-payoff automaton expression. In: ICALP. LNCS. Springer

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.