Approximating the minimum cycle mean[☆],[☆☆]

Krishnendu Chatterjee[a], Monika Henzinger[b], Sebastian Krinninger[b], Veronika Loitzenbauer[b,*], Michael A. Raskin[c]

[a]*Institute of Science and Technology, Am Campus 1, 3400 Klosterneuburg, Austria*
[b]*University of Vienna, Faculty of Computer Science, Währinger Straße 29, 1090 Vienna, Austria*
[c]*Independent University of Moscow, Bolshoy Vlasyevskiy 11, 115162 Moscow, Russia and Moscow Institute of Physics and Technology, 9 Institutskiy per., 141700 Dolgoprudny, Russia*

Abstract

We consider directed graphs where each edge is labeled with an integer weight and study the fundamental algorithmic question of computing the value of a cycle with minimum mean weight. Our contributions are twofold: (1) First we show that the algorithmic question is reducible to the problem of a logarithmic number of *min-plus* matrix multiplications of $n \times n$-matrices, where n is the number of vertices of the graph. (2) Second, when the weights are nonnegative, we present the first $(1+\epsilon)$-approximation algorithm for the problem and the running time of our algorithm is $\widetilde{O}(n^\omega \log^3 (nW/\epsilon)/\epsilon)$[1], where $O(n^\omega)$ is the time required for the *classic* $n \times n$-matrix multiplication and W is the maximum value of the weights. With an additional $O(\log(nW/\epsilon))$ factor in space a cycle with approximately optimal weight can be computed within the same time bound.

Keywords: quantitative verification, graph algorithm, mean-payoff objective, approximation algorithm

1. Introduction

Minimum cycle mean problem. We consider the fundamental algorithmic problem of computing the value of a minimum mean-weight cycle in a finite directed graph. The input to the problem is a directed graph $G = (V, E, w)$ with a finite set V of n vertices, a finite set E of m edges, and a weight function w that assigns an integer weight to every edge. Given a cycle C, the mean weight $\mu(C)$ of the cycle is the ratio of the sum of the weights of the cycle and the number of edges in the cycle. The algorithmic question asks to compute $\mu = \min\{\mu(C) \mid C \text{ is a cycle}\}$: the minimum cycle mean. The minimum cycle mean problem is an important problem in combinatorial optimization and has a long history of algorithmic study. An $O(nm)$-time algorithm for the problem was given by Karp [2]. The current best known algorithm for the problem by Orlin and Ahuja, which is over two decades old, requires $O(m\sqrt{n}\log(nW))$ time [3], where W is the maximum absolute value of the weights.

Applications. The minimum cycle mean problem is a basic combinatorial optimization problem that has numerous applications in network flows [4]. In the context of formal analysis of reactive systems, the performance of systems as well as the average resource consumption of systems is modeled as the minimum

[1]The $\widetilde{O}$-notation hides a polylogarithmic factor.

[☆]A preliminary version appeared in [1].

[*]Corresponding author. T.: +43-1-4277-78320
Email address: `veronika.loitzenbauer@univie.ac.at` (Veronika Loitzenbauer)

cycle mean problem. A reactive system is modeled as a directed graph, where vertices represent states of the system, edges represent transitions, and every edge is assigned a *nonnegative* integer representing the resource consumption (or delay) associated with the transition. The computation of a minimum average resource consumption behavior (or minimum average response time) corresponds to the computation of the minimum cycle mean. Several recent works model other quantitative aspects of system analysis (such as robustness) also as the mean-weight problem (also known as *mean-payoff* problem) [5, 6].

Results. This work contains the following results.

1. *Reduction to min-plus matrix multiplication.* We show that the minimum cycle mean problem is reducible to the problem of a logarithmic number of min-plus matrix multiplications of $n \times n$-matrices, where n is the number of vertices of the graph. Our result implies that algorithmic improvements for min-plus matrix multiplication will carry over to the minimum cycle mean problem with a logarithmic multiplicative factor in the running time.
2. *Faster approximation algorithm.* When the weights are nonnegative, we present the first $(1+\epsilon)$-approximation algorithm for the problem that outputs $\hat{\mu}$ such that $\mu \leq \hat{\mu} \leq (1+\epsilon)\mu$ and the running time of our algorithm is $\widetilde{O}(n^{\omega} \log^3 (nW/\epsilon)/\epsilon)$. As usual, the $\widetilde{O}$-notation is used to "hide" a polylogarithmic factor, i.e., $\widetilde{O}(T(n,m,W)) = O(T(n,m,W) \cdot \text{polylog}(n))$, and $O(n^{\omega})$ is the time required for the *classic* $n \times n$-matrix multiplication. The current best known bound for ω is $\omega < 2.3727$ [7, 8].
   For the computation of $\hat{\mu}$, $O(n^2)$ space is needed. If $O(n^2 \log(nW/\epsilon))$ space is used instead, i.e., the intermediate results of the approximation algorithm are saved, we can additionally output a cycle with mean weight at most $\hat{\mu}$.
   The worst case complexity of the current best known algorithm for the minimum cycle mean problem is $O(m\sqrt{n}\log(nW))$ [3], which could be as bad as $O(n^{2.5}\log(nW))$. Thus for $(1+\epsilon)$-approximation our algorithm provides better dependence on n.
   Note that in applications related to systems analysis the weights are always nonnegative (they represent resource consumption, delays, etc); and the weights are typically small, whereas the state space of the system is large. Moreover, due to imprecision in modeling, approximations in weights are already introduced during the modeling phase. Hence $(1+\epsilon)$-approximation of the minimum cycle mean problem with small weights and large graphs is a relevant algorithmic problem for reactive system analysis, and we improve the long-standing complexity of the problem.
   The key technique that we use to obtain the approximation algorithm is a combination of the value iteration algorithm for the minimum cycle mean problem, and a technique used for an approximation algorithm for all-pair shortest path problem for directed graphs. Table 1 compares our algorithm with the asymptotically fastest existing algorithms.

Reference	**Running time**	**Approximation**	**Range**
Karp [2]	$O(mn)$	exact	$[-W, W]$
Orlin and Ahuja [3]	$O(m\sqrt{n}\log(nW))$	exact	$[-W, W] \cap \mathbb{Z}$
Sankowski [9] (implicit)	$\widetilde{O}(Wn^{\omega}\log(nW))$	exact	$[-W, W] \cap \mathbb{Z}$
Butkovic and Cuninghame-Green [10]	$O(n^2)$	exact	$\{0, 1\}$
This paper	$\widetilde{O}(n^{\omega}\log^3(nW/\epsilon)/\epsilon)$	$1+\epsilon$	$[0, W] \cap \mathbb{Z}$

Table 1: Current fastest asymptotic running times for computing the minimum cycle mean

Outline. In the rest of this section we discuss related work and motivate the minimum cycle mean problem by its relation to negative cycle detection. We summarize all needed definitions in Section 2. In Section 3 we describe how min-plus matrix multiplication can be used to compute the minimum cycle mean exactly. In Section 4 we present our approximation algorithm and prove its correctness and running time. In Section 5 we show how at the cost of storing the intermediate results an approximately optimal cycle can be output.

1.1. Related work

The minimum cycle mean problem is basically equivalent to solving a deterministic Markov decision process (MDP) [11]. The latter can also be seen as a single-player mean-payoff game [12, 13, 11]. We distinguish two types of algorithms: algorithms that are independent of the weights of the graph and algorithms that depend on the weights in some way. By W we denote the maximum absolute edge weight of the graph.

Algorithms independent of weights. The classic algorithm of Karp [2] uses a dynamic programming approach to find the minimum cycle mean and runs in time $O(mn)$. A corresponding cycle can easily be computed given the outcome of the algorithm. The main drawback of Karp's algorithm is that its best-case and worst-case running times are the same. The algorithms of Hartmann and Orlin [14] and of Dasdan and Gupta [15] address this issue, but also have a worst-case complexity of $O(mn)$. By solving the more general parametric shortest path problem, Karp and Orlin [16] can compute the minimum cycle mean in time $O(mn \log n)$. Young, Tarjan, and Orlin [17] improve this running time to $O(mn + n^2 \log n)$.

A well known algorithm for solving MDPs is the value iteration algorithm. In each iteration this algorithm spends time $O(m)$ and in total it performs $O(nW)$ iterations. Madani [18] showed that, for *deterministic* MDPs (i.e., weighted graphs for which we want to find the minimum cycle mean), a certain variant of the value iteration algorithm "converges" to the optimal cycle after $O(n^2)$ iterations which gives a running time of $O(mn^2)$ for computing the minimum cycle mean. Using similar ideas he also obtains a running time of $O(mn)$. Howard's policy iteration algorithm is another well-known algorithm for solving MDPs [19]. The complexity of this algorithm for deterministic MDPs is unresolved. Recently, Hansen and Zwick [20] provided a class of weighted graphs on which Howard's algorithm performs $\Omega(n^2)$ iterations where each iteration takes time $O(m)$.[2]

Algorithms depending on weights. If a graph is complete and has only two different edge weights, then the minimum cycle mean problem can be solved in time $O(n^2)$ because the matrix of its weights is bivalent [10].

Another approach is to use the connection to the problem of detecting a negative cycle. Lawler [22] gave a reduction for finding the minimum cycle mean that performs $O(\log(nW))$ calls to a negative cycle detection algorithm. The main idea is to perform binary search on the minimum cycle mean. In each search step the negative cycle detection algorithm is run on a graph with modified edge weights. Orlin and Ahuja [3] extend this idea by the approximate binary search technique [23]. By combining approximate binary search with their scaling algorithm for the assignment problem they can compute the minimum mean cycle in time $O(m\sqrt{n} \log nW)$.

Note that in its full generality the single-source shortest paths problem (SSSP) also demands the detection of a negative cycle reachable from the source vertex.[3] Therefore it is also possible to reduce the minimum cycle mean problem to SSSP. The best time bounds on SSSP are as follows. Goldberg's scaling algorithm [24] solves the SSSP problem (and therefore also the negative cycle detection problem) in time $O(m\sqrt{n} \log W)$. McCormick [25] combines approximate binary search with Goldberg's scaling algorithm to find the minimum cycle mean in time $O(m\sqrt{n} \log nW)$, which matches the result of Orlin and Ahuja [3]. Sankowski's matrix multiplication based algorithm [9] solves the SSSP problem in time $\widetilde{O}(Wn^\omega)$. By combining binary search with Sankowski's algorithm, the minimum cycle mean problem can be solved in time $\widetilde{O}(Wn^\omega \log nW)$

Approximation of minimum cycle mean. To the best of our knowledge, our algorithm is the first approximation algorithm specifically for the minimum cycle mean problem. There are both additive and multiplicative fully polynomial-time approximation schemes for solving mean-payoff games [26, 27], which is a more general problem. Note that in contrast to finding the minimum cycle mean it is not known whether the exact solution to a mean-payoff game can be computed in polynomial time. The results of [26] and [27] are obtained by reductions to a pseudo-polynomial algorithm for solving mean-payoff games. In the case of

[2]See [21] for a summary of recent results on Howard's algorithm.

[3]Remember that, for example, Dijkstra's algorithm for computing single-source shortest paths requires nonnegative edge weights which excludes the possibility of negative cycles.

the minimum cycle mean problem, these reductions do not provide an improvement over the current fastest exact algorithms mentioned above.

Min-plus matrix multiplication. Our approach reduces the problem of finding the minimum cycle mean to computing the (approximate) min-plus product of matrices. The naive algorithm for computing the min-plus product of two matrices runs in time $O(n^3)$. To date, no algorithm is known that runs in time $O(n^{3-\alpha})$ for some $\alpha > 0$, so-called *truly subcubic* time. This is in contrast to classic matrix multiplication that can be done in time $O(n^\omega)$ where the current best bound on ω is $\omega < 2.3727$ [7, 8]. Moreover, Vassilevska Williams and Williams [28] showed that computing the min-plus product is computationally "equivalent" to a series of problems including all-pairs shortest paths and negative triangle detection in the following sense: if one of these problems has a truly subcubic algorithm, then all of them have. This provides evidence for the hardness of these problems.

Still, the running time of $O(n^3)$ for the min-plus product can be improved. Fredman [29] gave an algorithm for computing the min-plus product with a slightly subcubic running time of $O(n^3(\log\log n)^{1/3}/(\log n)^{1/3})$. After a long line of improvements, Chan [30] presented an algorithm of similar flavor with a running time of $O(n^3(\log\log n)^3/(\log n)^2)$. Very recently Williams [31] developed a randomized algorithm that runs in time $O(n^3/2^{\Omega(\log n/\log\log n)^{1/2}})$ and is correct with high probability. Williams also gave a deterministic version that runs in time $O(n^3/2^{\log^\delta n})$ for some $\delta > 0$.

A different approach for computing the min-plus product of two integer matrices is to reduce the problem to classic matrix multiplication [32]. In this way, the min-plus product can be computed in time $O(Mn^\omega \log M)$ which is pseudo-polynomial since M is the maximum absolute integer entry [33]. This observation was used by Alon, Galil, and Margalit [33] and Zwick [34] to obtain faster all-pairs shortest paths algorithms in directed graphs for the case of small integer edge weights. Zwick also combines this min-plus matrix multiplication algorithm with an adaptive scaling technique that allows to compute $(1+\epsilon)$-approximate all-pairs shortest paths in graphs with nonnegative edge weights. Our approach of finding the minimum cycle mean extensively uses this technique.

1.2. Relation to negative cycle detection

In the following we provide additional motivation for our approach of *approximating* the minimum cycle mean by relating it to negative cycle detection. A solution to the minimum cycle mean problem immediately gives a solution to the negative cycle detection problem. Therefore, an improved running time for finding the minimum cycle mean will also give an improved running time for detecting a negative cycle, which in turn has numerous applications [35] and comes up as a subproblem in algorithms for other problems, such as the minimum-cost flow problem [36]. However, researchers are stuck with finding faster algorithms for negative cycle detection. Even more, by the binary-search based reduction of minimum cycle mean to negative cycle detection, the worst-case running times of both problems are the same, up to a factor of $O(\log nW)$. Approximation helps to break the running time barrier induced by negative cycle detection. Intuitively, the approximation provided by our algorithm is not good enough to distinguish between a positive and a negative cycle. Therefore our approximation algorithm can be faster than known algorithms for the negative cycle detection problem.

Our new algorithm needs two non-standard assumptions. First, it only works for graphs with nonnegative edge weights. Second, it provides a multiplicative approximation. Both of these assumptions are necessary for bypassing the negative cycle detection problem—only one of them is not enough. On the one hand, if we could compute the minimum cycle mean exactly for nonnegative edge weights, then, by shifting of weights, we could solve the negative cycle detection problem. On the other hand, a reduction of Gentilini [37] (initially designed for mean-payoff games) shows that if one can compute a multiplicative ϵ-approximation $\hat{\mu}$ of the minimum cycle mean μ such that $|(\hat{\mu}-\mu)/\mu| \leq \epsilon$, then one can immediately detect negative cycles. In particular, this is true for $\epsilon \leq 1$, for which μ and $\hat{\mu}$ will always have the same sign. To see this, note that when $\hat{\mu}$ and μ have different signs, then $|(\hat{\mu}-\mu)/\mu| = (|\hat{\mu}|+|\mu|)/|\mu| = |\hat{\mu}|/|\mu| + 1 > 1$. Thus, the only hope of getting a multiplicative ϵ-approximation for arbitrary edge weights without solving the negative cycle detection problem is when $\epsilon > 1$. We remark that Gentilini's definition of an ϵ-approximation is a generalization of our definition of a $(1+\epsilon)$-approximation to arbitrary edge weights.

2. Definitions

Throughout this paper we let $G = (V, E, w)$ be a weighted directed graph with a finite set of vertices V and a set of edges E such that every vertex has at least one outgoing edge. The weight function w assigns a nonnegative integer weight to every edge. We denote by n the number of vertices of G and by m the number of edges of G. Note that $m \geq n$ because every vertex has at least one outgoing edge.

A *path* is a finite sequence of edges $P = (e_1, \ldots, e_k)$ such that for all consecutive edges $e_i = (x_i, y_i)$ and $e_{i+1} = (x_{i+1}, y_{i+1})$ of P we have $y_i = x_{i+1}$. Note that edges may be repeated on a path, we *do not* only consider simple paths. The *length* $|P|$ *of a path* $P = (e_1, \ldots, e_k)$ is the number of edges of P, i.e. $|P| = k$. The *weight of a path* $P = (e_1, \ldots, e_k)$, denoted by $w(P)$, is the sum of its edge weights, i.e. $w(P) = \sum_{1 \leq i \leq k} w(e_i)$. The *mean weight* of the path P is the ratio $w(P)/|P|$. A *cycle* is a path in which the start vertex and the end vertex are the same. In a *simple cycle* each vertex contained in the cycle appears in exactly two of the edges of the cycle; thus the length of a simple cycle can be at most n.

The *minimum cycle mean* of G is the minimum mean weight of any cycle in G. For every vertex x we denote by $\mu(x)$ the value of the minimum mean-weight cycle reachable from x. The minimum cycle mean of G is simply the minimum $\mu(x)$ over all vertices x.

We will use that there always exists a *simple* cycle with minimum mean weight and thus we can assume that the cycle with minimum mean weight has at most n edges. This is already used implicitly in [2]. We show first that every non-simple cycle can be partitioned into a set of simple cycles, which already appeared in [11].

Proposition 1 ([11])**.** *Let P be a path in the graph $G = (V, E)$ from x to y. Let $G_P = (V, E_P)$ be the multigraph consisting of the edges in P. Then E_P can be partitioned into a simple path from x to y and a set S of simple cycles.*

Proof. Initialize the set S with the empty set. Follow the path P until you encounter a vertex for the second time. Let v be this vertex and let C be the set of edges between the first and the second encounter of v. Then C is a simple cycle since no other vertex was encountered twice. Add C to S, remove C from P and follow the updated path P', starting again from vertex x, until you encounter a vertex for the second time. Repeat this removal of simple cycles until the final vertex of P is reached without encountering any vertex twice. Then the remaining path is a simple path from x to y. $\square$

Proposition 2. *Given a set of simple cycles S with total mean weight $\mu = \sum_{C_i \in S} \sum_{e \in C_i} w(e) / \sum_{C_i \in S} |C_i|$, there exists a simple cycle in S with mean weight at most μ.*

Proof. Denote for each simple cycle $C_i \in S$ its mean weight by μ_i and its number of edges by m_i. Then

$$\mu \sum_{C_i \in S} m_i = \frac{\sum_{C_i \in S} \sum_{e \in C_i} w(e)}{\sum_{C_i \in S} m_i} \sum_{C_i \in S} m_i = \sum_{C_i \in S} w(C_i) = \sum_{C_i \in S} \mu_i m_i \geq \min_{C_i \in S}(\mu_i) \cdot \sum_{C_i \in S} m_i$$

and thus $\min_{C_i \in S}(\mu_i) \leq \mu$. $\square$

Corollary 3. *Let μ be the minimum cycle mean of a graph G. Then there exists a* simple *cycle in G with mean weight μ.*

For every vertex x and every integer $t \geq 1$ we denote by $\delta_t(x)$ the minimum weight of all paths starting at x that have length t, i.e., consist of exactly t edges. For all pairs of vertices x and y and every integer $t \geq 1$ we denote by $d_t(x, y)$ the minimum weight of all paths of length t from x to y. If no such path exists we set $d_t(x, y) = \infty$.

For every matrix A we denote by $A[i, j]$ the entry at the i-th row and the j-th column of A. We only consider $n \times n$ matrices with integer entries, where n is the size of the graph. We assume that the vertices of G are numbered consecutively from 1 to n, which allows us to use $A[x, y]$ to refer to the entry of A belonging to vertices x and y. The *weight matrix D of G* is the matrix containing the weights of G. For all pairs of vertices x and y we set $D[x, y] = w(x, y)$ if the graph contains the edge (x, y) and $D[x, y] = \infty$ otherwise.

We denote the *min-plus product* of two matrices A and B by $A \star B$. The min-plus product is defined as follows. If $C = A \star B$, then for all indices $1 \leq i, j \leq n$ we have $C[i,j] = \min_{1 \leq k \leq n}(A[i,k] + B[k,j])$. We denote by A^t the t-th power of the matrix A. Formally, we set $A^1 = A$ and $A^{t+1} = A \star A^t$ for $t \geq 1$. We denote by ω the exponent of classic matrix multiplication, i.e., the product of two $n \times n$ matrices can be computed in time $O(n^\omega)$. The current best bound on ω is $\omega < 2.3727$ [7, 8].

3. Reduction of minimum cycle mean to min-plus matrix multiplication

In the following we explain the main idea of our approach which is to use min-plus matrix multiplication to find the minimum cycle mean. The well-known value iteration algorithm uses a dynamic programming approach to compute in each iteration a value for every vertex x from the values of the previous iteration. After t iterations, the value computed by the value iteration algorithm for vertex x is equal to $\delta_t(x)$, the minimum weight of all paths with length t starting at x. We are actually interested in $\mu(x)$, the value of the minimum mean-weight cycle reachable from x. It is well known that $\lim_{t \to \infty} \delta_t(x)/t = \mu(x)$ and that the value of $\mu(x)$ can be computed from $\delta_t(x)$ if t is large enough ($t = O(n^3 W)$) [11].[4] Thus, one possibility to determine $\mu(x)$ is the following: first, compute $\delta_t(x)$ for t large enough with the value iteration algorithm and then compute $\mu(x)$ from $\delta_t(x)$. However, using the value iteration algorithm for computing $\delta_t(x)$ is expensive because its running time is linear in t and thus pseudo-polynomial.

Our idea is to compute $\delta_t(x)$ for a large value of t by using fast matrix multiplication instead of the value iteration algorithm. We will compute the matrix D^t, the t-th power of the weight matrix (using min-plus matrix multiplication). The matrix D^t contains the value of the minimum-weight path of length exactly t for all pairs of vertices. Given D^t, we can determine the value $\delta_t(x)$ for every vertex x by finding the minimum entry in the row of D^t corresponding to x.

Proposition 4. *For every $t \geq 1$ and all vertices x and y we have (i) $d_t(x,y) = D^t[x,y]$ and (ii) $\delta_t(x) = \min_{y \in V} D^t[x,y]$.*

Proof. We give the proof for the sake of completeness. The claim $d_t(x,y) = D^t[x,y]$ follows from a simple induction on t. If $t = 1$, then clearly the minimal-weight path of length 1 from x to y is the edge from x to y if it exists, otherwise $d_t(x,y) = \infty$. If $t \geq 1$, then a minimal-weight path of length t from x to y (if it exists) consists of some outgoing edge of $e = (x,z)$ as its first edge and then a minimal-weight path of length $t-1$ from z to y. We therefore have $d_t(x,y) = \min_{(x,z) \in E} w(x,z) + d_{t-1}(z,y)$. By the definition of the weight matrix and the induction hypothesis we get $d_t(x,y) = \min_{z \in V} D[x,z] + D^{t-1}[z,y]$. Therefore the matrix $D \star D^{t-1} = D^t$ contains the value of $d_t(x,y)$ for every pair of vertices x and y.

For the second claim, $\delta_t(x) = \min_{y \in V} D^t[x,y]$, observe that by the definition of $\delta_t(x)$ we obviously have $\delta_t(x) = \min_{y \in V} d_t(x,y)$ because the minimal-weight path of length t starting at x has *some* node y as its end point. □

Using this approach, the main question is how fast the matrix D^t can be computed. The most important observation is that D^t (and therefore also $\delta_t(x)$) can be computed by repeated squaring with only $O(\log t)$ min-plus matrix multiplications. This is different from the value iteration algorithm, where t iterations are necessary to compute $\delta_t(x)$.

Proposition 5. *For every $t \geq 1$ we have $D^{2t} = D^t \star D^t$. Therefore the matrix D^t can be computed with $O(\log t)$ many min-plus matrix multiplications.*

Proof. We give the proof for the sake of completeness. It can easily be verified that the min-plus matrix product is associative [38] and therefore $D^{2t} = D^t \star D^t$. Therefore, if t is a power of two, we can compute D^t with $\log t$ min-plus matrix multiplications. If t is not a power of two, we can decompose D^t into $D^t = D^{t_1} \star \ldots \star D^{t_k}$ where each $t_i \leq t$ (for $1 \leq i \leq k$) is a power of two and $k \leq \lceil \log t \rceil$. By storing intermediate results, we can

[4] Specifically, for $t = 4n^3 W$ the unique number in $(\delta_t(x)/t - 1/[2n(n-1)], \delta_t(x)/t + 1/[2n(n-1)]) \cap \mathbb{Q}$ that has a denominator of at most n is equal to $\mu(x)$ [11].

compute D^{2^i} for every $0 \leq i \leq \lceil \log t \rceil$ with $\lceil \log t \rceil$ min-plus matrix multiplications. Using the decomposition above, we have to multiply at most $\lceil \log t \rceil$ such matrices to obtain D^t. Therefore the total number of min-plus matrix multiplications needed for computing D^t is $O(\log t)$. $\square$

The running time of this algorithm depends on the time needed for computing the min-plus product of two integer matrices. This running time will usually depend on the two parameters n and M where n is the size of the $n \times n$ matrices to be multiplied (in our case this is equal to the number of vertices of the graph) and the parameter M denotes the maximum absolute integer entry in the matrices to be multiplied. When we multiply the matrix D by itself to obtain D^2, we have $M = W$, where W is the maximum absolute edge weight. However, M increases with every multiplication and in general, we can bound the maximum absolute integer entry of the matrix D^t only by $M = tW$. Note that $O(n^2)$ operations are necessary to extract the minimum cycle mean $\mu(x)$ for all vertices x from the matrix D^t (with $t = O(n^3 W)$ [11], see above).

Theorem 6. *If the min-plus product of two $n \times n$ matrices with entries in $\{-M, \ldots, -1, 0, 1, \ldots, M, \infty\}$ can be computed in time $T(n, M)$, then the minimum cycle mean problem can be solved in time $T(n, tW) \log t$ where $t = O(n^3 W)$.*[5]

Unfortunately, the approach outlined above does not immediately improve the running time for the minimum cycle mean problem because min-plus matrix multiplication currently cannot be done fast enough. However, our approach is still useful for solving the minimum cycle mean problem *approximately* because approximate min-plus matrix multiplication can be done faster than its exact counterpart.

4. Approximation algorithm

In this section we design an algorithm that computes an approximation of the minimum cycle mean in graphs with nonnegative integer edge weights. It follows the approach of reducing the minimum cycle mean problem to min-plus matrix multiplication outlined in Section 3. The key to our algorithm is a fast procedure for computing the min-plus product of two integer matrices approximately. We will proceed as follows. First, we explain how to compute an approximation F of D^t, the t-th power of the weight matrix D. From this we easily get, for every vertex x, an approximation $\hat{\delta}_t(x)$ of $\delta_t(x)$, the minimum-weight of all paths of length t starting at x. We then argue that for t large enough (in particular $t = O(n^2 W/\epsilon)$), the value $\delta_t(x)/t$ is an approximation of $\mu(x)$, the minimum cycle mean of cycles reachable from x. By combining both approximations we can show that $\hat{\delta}_t(x)/t$ is an approximation of $\mu(x)$. Thus, the main idea of our algorithm is to compute an approximation of D^t for a large enough t.

4.1. Computing an approximation of D^t

Our first goal is to compute an approximation of the matrix D^t, the t-th power of the weight matrix D, given $t \geq 1$. Zwick provides the following algorithm for approximate min-plus matrix multiplication.

Theorem 7 (Zwick [34]). *Let A and B be two $n \times n$ matrices with integer entries in $[0, M]$ and let $C := A \star B$. Let $R \geq \log n$ be a power of two. The algorithm* approx-min-plus(A, B, M, R) *computes the approximate min-plus product $\overline{C}$ of A and B in time*[6] $O(n^\omega R \log(M) \log^2(R) \log(n))$ *such that for every $1 \leq i, j \leq n$ it holds that $C[i,j] \leq \overline{C}[i,j] \leq (1 + 4/R) C[i,j]$.*

We now give a modification (see Algorithm 1) of Zwick's algorithm for approximate shortest paths [34] such that, given an ϵ in $(0, 1]$, the algorithm computes a $(1+\epsilon)$-approximation F of D^t when t is a power of two such that for $1 \leq i, j \leq n$ we have $D^t[i,j] \leq F[i,j] \leq (1+\epsilon) D^t[i,j]$. Just as we can compute D^t exactly

[5]Note that necessarily $T(n, M) = \Omega(n^2)$ because the result matrix has n^2 entries that have to be written.

[6]The running time of approx-min-plus is given by $O(n^\omega \log M)$ times the time needed to multiply two $O(R \log n)$-bit integers. With the Schönhage-Strassen algorithm for large integer multiplication, two k-bit integers can be multiplied in $O(k \log k \log \log k)$ time, which gives a running time of $O(n^\omega R \log(M) \log(n) \log(R \log n) \log \log(R \log n))$. This can be bounded by the running time given in Theorem 7 if $R \geq \log n$, which will always be the case in the following.

with $\log t$ min-plus matrix multiplications, the algorithm computes the $(1+\epsilon)$-approximation of D^t in $\log t$ iterations. However, in each iteration only an approximate min-plus product is computed. Let F_s be the approximation of $D_s := D^{2^s}$. In the s-th iteration we use approx-min-plus$(F_{s-1}, F_{s-1}, tW, R)$ to calculate F_s with R chosen beforehand such that the desired error bound is reached for $F = F_{\log t}$.

Algorithm 1: Approximation of D^t

```
input  : weight matrix $D$, error bound $\epsilon \in (0,1]$, $t$ (a power of 2)
output : $(1+\epsilon)$-approximation of $D^t$
$F \leftarrow D$
$r \leftarrow 4\log t/\ln(1+\epsilon)$
$R \leftarrow 2^{\lceil \log r \rceil}$
for $\log t$ times do
    $F \leftarrow$ approx-min-plus$(F, F, 2tW, R)$
end
return $F$
```

Lemma 8. *Given an $0 < \epsilon \leq 1$ and a power of two $t \geq 1$, Algorithm 1 computes a $(1+\epsilon)$-approximation F of D^t in time*

$$O\left(n^\omega \cdot \frac{\log^2(t)}{\epsilon} \cdot \log(tW) \log^2\left(\frac{\log(t)}{\epsilon}\right) \log(n)\right) = \widetilde{O}\left(n^\omega \cdot \frac{\log^2(t)}{\epsilon} \cdot \log(tW)\right)$$

such that $D^t[i,j] \leq F[i,j] \leq (1+\epsilon)\, D^t[i,j]$ for all $1 \leq i,j \leq n$.

Proof. We start with the main idea of the proof and continue with the details afterwards. The running time of approx-min-plus depends linearly on R and logarithmically on M, the maximum entry of the input matrices. Algorithm 1 calls approx-min-plus $\log t$ times. Each call increases the error by a factor of $(1+4/R)$. However, as only $\log t$ approximate matrix multiplications are used, setting R to the smallest power of 2 that is larger than $4\log(t)/\ln(1+\epsilon)$ will suffice to bound the approximation error by $(1+\epsilon)$. We will show that $2tW$ is an upper bound on the entries in the input matrices for approx-min-plus. The stated running time will follow from these two facts and Theorem 7.

Let F_s be the approximation of $D_s := D^{2^s}$ computed by the algorithm after iteration s. Recall that $2^s W$ is an upper bound on the maximum entry in D_s. As we will show, all entries in F_s are at most $(1+\epsilon)$-times the entries in D_s. Since we assume $\epsilon \leq 1$, we have $1+\epsilon \leq 2$. Thus $2^{s+1}W$ is an upper bound on the entries in F_s. Hence $2tW$ is an upper bound on the entries of F_s with $1 \leq s < \log t$, i.e., for all input matrices of approx-min-plus in our algorithm.

This results in an overall running time of

$$\begin{aligned} &O\left(n^\omega R \log(tW) \log^2(R) \log(n) \cdot \log(t)\right) \\ &= O\left(n^\omega \cdot \frac{\log^2(t)}{\log(1+\epsilon)} \cdot \log(tW) \log^2\left(\frac{\log(t)}{\log(1+\epsilon)}\right) \log(n)\right) \\ &= O\left(n^\omega \cdot \frac{\log^2(t)}{\epsilon} \cdot \log(tW) \log^2\left(\frac{\log(t)}{\epsilon}\right) \log(n)\right) . \end{aligned}$$

The last equation follows from the inequality $\ln(x) \leq x - 1$ for $x > 0$: With $x = 1/(1+\epsilon)$ and $\epsilon > 0$ we have $1/\ln(1+\epsilon) \leq (1+\epsilon)/\epsilon$. Since $\epsilon \leq 1$ it follows that $1/\log(1+\epsilon) = O(1/\epsilon)$.

To show the claimed approximation guarantee, we will prove that the inequality

$$D_s[i,j] \leq F_s[i,j] \leq \left(1 + \frac{4}{R}\right)^s D_s[i,j] \tag{1}$$

holds after the s-th iteration of Algorithm 1 by induction on s. Note that the $(1+\epsilon)$-approximation follows from this inequality because the parameter R is chosen such that after the $(\log t)$-th iteration of the algorithm it holds that

$$\left(1+\frac{4}{R}\right)^{\log t} \leq \left(1+\frac{\ln(1+\epsilon)}{\log t}\right)^{\log t} \leq e^{\ln(1+\epsilon)} = 1+\epsilon\,. \tag{2}$$

For $s=0$ we have $F_s = D_s$ and the inequality holds trivially. Assume the inequality holds for s. We will show that it also holds for $s+1$.

First we prove the lower bound on $F_{s+1}[i,j]$. Let C_{s+1} be the exact min-plus product of F_s with itself, i.e., $C_{s+1} = F_s \star F_s$. Let k_c be the minimizing index such that $C_{s+1}[i,j] = \min_{1\leq k\leq n}(F_s[i,k]+F_s[k,j]) = F_s[i,k_c]+F_s[k_c,j]$. By the definition of the min-plus product

$$D_{s+1}[i,j] = \min_{1\leq k\leq n}(D_s[i,k]+D_s[k,j]) \leq D_s[i,k_c]+D_s[k_c,j]\,. \tag{3}$$

By the induction hypothesis and the definition of k_c we have

$$D_s[i,k_c]+D_s[k_c,j] \leq F_s[i,k_c]+F_s[k_c,j] = C_{s+1}[i,j]\,. \tag{4}$$

By Theorem 7 the values of F_{s+1} can only be larger than the values in C_{s+1}, i.e.,

$$C_{s+1}[i,j] \leq F_{s+1}[i,j]\,. \tag{5}$$

Combining inequalities (3), (4), and (5) yields the claimed lower bound,

$$D_{s+1}[i,j] \leq F_{s+1}[i,j]\,.$$

Next we prove the upper bound on $F_{s+1}[i,j]$. Let k_d be the minimizing index such that $D_{s+1}[i,j] = D_s[i,k_d]+D_s[k_d,j]$. Theorem 7 gives the error from one call of approx-min-plus, i.e., the error in the entries of F_{s+1} compared to the entries of C_{s+1}. We have

$$F_{s+1}[i,j] \leq \left(1+\frac{4}{R}\right) C_{s+1}[i,j]\,. \tag{6}$$

By the definition of the min-plus product we know that

$$C_{s+1}[i,j] \leq F_s[i,k_d]+F_s[k_d,j]\,. \tag{7}$$

By the induction hypothesis and the definition of k_d we can reformulate the error obtained in the first s iterations of Algorithm 1 as follows:

$$\begin{aligned} F_s[i,k_d]+F_s[k_d,j] &\leq \left(1+\frac{4}{R}\right)^s D_s[i,k_d] + \left(1+\frac{4}{R}\right)^s D_s[k_d,j] \\ &= \left(1+\frac{4}{R}\right)^s (D_s[i,k_d]+D_s[k_d,j]) \\ &= \left(1+\frac{4}{R}\right)^s D_{s+1}[i,j]\,. \end{aligned} \tag{8}$$

Combining inequalities (6), (7), and (8) yields the upper bound

$$F_{s+1}[i,j] \leq \left(1+\frac{4}{R}\right)^{s+1} D_{s+1}[i,j]\,. \qquad \square$$

Once we have computed an approximation of the matrix D^t, we extract from it the minimal entry of each row to obtain an approximation of $\delta_t(x)$. Here we use the equivalence between the minimum entry of row x of D^t and $\delta_t(x)$ established in Proposition 4. Remember that $\delta_t(x)/t$ approaches $\mu(x)$ for t large enough and later on we want to use the approximation of $\delta_t(x)$ to obtain an approximation of the minimum cycle mean $\mu(x)$.

Lemma 9. *The value $\hat{\delta}_t(x) := \min_{y \in V} F[x,y]$ approximates $\delta_t(x)$ with $\delta_t(x) \leq \hat{\delta}_t(x) \leq (1+\epsilon)\delta_t(x)$.*

Proof. Let y_f and y_d be the indices where the x-th rows of F and D^t obtain their minimal values, respectively, i.e.,

$$y_f := \arg\min_{y \in V} F[x,y] \quad \text{and} \quad y_d := \arg\min_{y \in V} D^t[x,y]\,.$$

By these definitions and Lemma 8 we have

$$\delta_t(x) = D^t[x,y_d] \leq D^t[x,y_f] \leq F[x,y_f] = \hat{\delta}_t(x)$$

and

$$\hat{\delta}_t(x) = F[x,y_f] \leq F[x,y_d] \leq (1+\epsilon)D^t[x,y_d]\,.$$ □

4.2. Approximating the minimum cycle mean

We now add the next building block to our algorithm. So far, we can obtain an approximation $\hat{\delta}_t(x)$ of $\delta_t(x)$ for any t that is a power of two. We now show that $\delta_t(x)/t$ is itself an approximation of the minimum cycle mean $\mu(x)$ for t large enough. Then we argue that $\hat{\delta}_t(x)/t$ approximates the minimum cycle mean $\mu(x)$ for t large enough. This value of t bounds the number of iterations of our algorithm. A similar technique was also used in [11] to bound the number of iterations of the value iteration algorithm for the two-player mean-payoff game.

We start by showing that $\delta_t(x)/t$ differs from $\mu(x)$ by at most nW/t *for any* t. Then we will turn this additive error into a multiplicative error by choosing a large enough value of t. A multiplicative error implies that we have to compute the solution exactly for $\mu(x) = 0$. We will use a separate procedure to identify all vertices x with $\mu(x) = 0$ and compute the approximation only for the remaining vertices. Note that $\mu(x) > 0$ implies $\mu(x) \geq 1/n$ because all edge weights are integers and we can assume by Corollary 3 that the cycle with minimum mean weight has at most n edges.

Lemma 10. *For every $x \in V$ and every integer $t \geq 1$ it holds that*

$$t \cdot \mu(x) - nW \leq \delta_t(x) \leq t \cdot \mu(x) + nW\,.$$

Proof. We first show the lower bound on $\delta_t(x)$. Let P be a path of length t starting at x with weight $\delta_t(x)$. Consider the cycles in P and let E' be the multiset of the edges in P that are in a cycle of P. There can be at most n edges that are not in a cycle of P, thus there are at least $\max(t-n, 0)$ edges in E'. Since $\mu(x)$ is the minimum mean weight of any cycle reachable from x, the sum of the weights of the edges in E' can be bounded below by $\mu(x)$ times the number of edges in E'. Furthermore, the value of $\mu(x)$ can be at most W. As we only allow nonnegative edge weights, the sum of the weights of the edges in E' is a lower bound on $\delta_t(x)$. Thus we have

$$\delta_t(x) \geq \sum_{e \in E'} w(e) \geq (t-n)\mu(x) \geq t \cdot \mu(x) - n \cdot \mu(x) \geq t \cdot \mu(x) - nW\,.$$

Next we prove the upper bound on $\delta_t(x)$. Let l be the length of the shortest path from x to a vertex y in a minimum mean-weight cycle C reachable from x (such that only y is both in the shortest path and in C). Let c be the length of C. By Corollary 3 we can assume that C is a simple cycle. Let the path Q be a path of length t that consists of the shortest path from x to y, $\lfloor (t-l)/c \rfloor$ rounds on C, and $t - l - c\lfloor (t-l)/c \rfloor$ additional edges in C. By the definition of $\delta_t(x)$, we have $\delta_t(x) \leq w(Q)$. The sum of the length of the shortest path from x to y and the number of the remaining edges of Q not in a complete round on C can be at most n because in a graph with nonnegative weights no shortest path has a cycle and no vertices in C except y are contained in the shortest path from x to y. Each of these edges has a weight of at most W. The mean weight of C is $\mu(x)$, thus the sum of the weights of the edges in all complete rounds on C is $\mu(x) \cdot c\lfloor (t-l)/c \rfloor \leq \mu(x) \cdot t$. Hence we have

$$\delta_t(x) \leq w(Q) \leq t \cdot \mu(x) + nW\,.$$ □

In the next step we show that we can use the fact that $\delta_t(x)/t$ is an approximation of $\mu(x)$ to obtain a $(1+\epsilon)$-approximation $\hat{\mu}(x)$ of $\mu(x)$ even if we only have an approximation $\hat{\delta}_t(x)$ of $\delta_t(x)$ with $(1+\epsilon)$-error. We exclude the case $\mu(x) = 0$ for the moment.

Lemma 11. *Assume we have an approximation $\hat{\delta}_t(x)$ of $\delta_t(x)$ such that $\delta_t(x) \leq \hat{\delta}_t(x) \leq (1+\epsilon)\delta_t(x)$ for $0 < \epsilon \leq 1/2$. If*

$$t \geq \frac{n^2 W}{\epsilon}\,, \quad \mu(x) \geq \frac{1}{n}\,, \quad \text{and} \quad \hat{\mu}(x) := \frac{\hat{\delta}_t(x)}{(1-\epsilon)t}\,,$$

then

$$\mu(x) \leq \hat{\mu}(x) \leq (1+7\epsilon)\mu(x)\,.$$

Proof. We first show that $\hat{\mu}(x)$ is at least as large as $\mu(x)$. From Lemma 10 we have $\delta_t(x) \geq t \cdot \mu(x) - nW$. As t is chosen large enough,

$$\frac{\delta_t(x)}{t} \geq \mu(x) - \frac{nW}{t} \geq \mu(x) - \frac{\epsilon}{n} \geq \mu(x) - \epsilon\mu(x) \geq (1-\epsilon)\mu(x)\,.$$

Thus, by the assumption $\delta_t(x) \leq \hat{\delta}_t(x)$ we have

$$\mu(x) \leq \frac{\hat{\delta}_t(x)}{(1-\epsilon)t} = \hat{\mu}(x)\,.$$

For the upper bound on $\hat{\mu}(x)$ we use the inequality $\delta_t(x) \leq t \cdot \mu(x) + nW$ from Lemma 10. As t is chosen large enough,

$$\frac{\delta_t(x)}{t} \leq \mu(x) + \frac{nW}{t} \leq \mu(x) + \frac{\epsilon}{n} \leq (1+\epsilon)\mu(x)\,. \tag{9}$$

With $\hat{\delta}_t(x) \leq (1+\epsilon)\delta_t(x)$ this gives

$$\hat{\mu}(x) = \frac{\hat{\delta}_t(x)}{(1-\epsilon)t} \leq \frac{(1+\epsilon)^2}{(1-\epsilon)}\mu(x)\,.$$

It can be verified by simple arithmetic that for $\epsilon > 0$ the inequality $\epsilon \leq 1/2$ is equivalent to

$$\frac{(1+\epsilon)^2}{(1-\epsilon)} \leq (1+7\epsilon)\,. \qquad \square$$

As a last ingredient to our approximation algorithm, we design a procedure that deals with the special case that the minimum cycle mean is 0. Since our goal is an algorithm with multiplicative error, we have to be able to compute the solution exactly in that case. This can be done in linear time because the edge-weights are nonnegative.

Proposition 12. *Given a graph with nonnegative integer edge weights, we can find all vertices x with $\mu(x) = 0$ and output a cycle with mean weight of 0 in time $O(m)$.*

Proof. Note that in the case of nonnegative edge weights we have $\mu(x) \geq 0$. Furthermore, a cycle can only have mean weight 0 if all edges on this cycle have weight 0. Thus, it will be sufficient to detect cycles in the graph that only contain edges that have weight 0.

We proceed as follows. Let $G^0 = (V, E^0)$ denote the subgraph of G that only contains edges of weight 0, i.e., $E^0 = \{e \in E | w(e) = 0\}$. As argued above, G contains a zero-mean cycle if and only if G^0 contains a cycle. We can check whether G^0 contains a cycle by computing the strongly connected components of G^0: G^0 contains a cycle if and only if it has a strongly connected component of size at least 2 (we can assume w.l.o.g. that there are no self-loops). Let Z be the set of all vertices in a strongly connected components of G^0 of size at least 2. We can identify all vertices that can reach a zero-mean cycle by performing a linear-time graph traversal to identify all vertices that can reach Z.

To actually output a zero-mean cycle, consider one of the strongly connected components of G^0 of size at least 2 and output a cycle found by a linear-time traversal of the component.

Since all steps take linear time, the total running time of this algorithm is $O(m)$. $\square$

Finally, we wrap up all arguments to obtain our algorithm for approximating the minimum cycle mean. This algorithms performs $\log t$ approximate min-plus matrix multiplications to compute an approximation of D^t and $\delta_t(x)$. Lemma 11 tells us that $t = n^2W/\epsilon$ is just the right number to guarantee that our approximation of $\delta_t(x)$ can be used to obtain an approximation of $\mu(x)$. The value of t is relatively large but the running time of our algorithm depends on t only in a logarithmic way.

Theorem 13. *Given a graph with nonnegative integer edge weights, we can compute an approximation $\hat{\mu}(x)$ of the minimum cycle mean for every vertex x such that $\mu(x) \le \hat{\mu}(x) \le (1+\epsilon)\mu(x)$ for $0 < \epsilon \le 1$ in time*

$$O\left(\frac{n^\omega}{\epsilon}\log^3\left(\frac{nW}{\epsilon}\right)\log^2\left(\frac{\log\left(\frac{nW}{\epsilon}\right)}{\epsilon}\right)\log(n)\right) = \widetilde{O}\left(\frac{n^\omega}{\epsilon}\log^3\left(\frac{nW}{\epsilon}\right)\right).$$

Proof. First we find all vertices x with $\mu(x) = 0$. By Proposition 12 this takes time $O(n^2)$ for $m = O(n^2)$. For the remaining vertices x we approximate $\mu(x)$ as follows.

Let $\epsilon' := \epsilon/7$. If we execute Algorithm 1 with weight matrix D, error bound ϵ' and t such that t is the smallest power of two with $t \ge n^2W/\epsilon'$, we obtain a $(1+\epsilon')$-approximation $F[x,y]$ of $D^t[x,y]$ for all vertices x and y (Lemma 8). By calculating for every x the minimum entry of $F[x,y]$ over all y we have a $(1+\epsilon')$-approximation of $\delta_t(x)$ (Lemma 9). By Lemma 11 $\hat{\mu}(x) := \hat{\delta}_t(x)/((1-\epsilon')t)$ is for this choice of t an approximation of $\mu(x)$ such that $\mu(x) \le \hat{\mu}(x) \le (1+7\epsilon')\mu(x)$. By substituting ϵ' with $\epsilon/7$ we get $\mu(x) \le \hat{\mu}(x) \le (1+\epsilon)\mu(x)$, i.e., a $(1+\epsilon)$-approximation of $\mu(x)$.

By Lemma 8 the running time of Algorithm 1 for $t = 2^{\lceil \log(n^2W/\epsilon')\rceil} = O(n^2W/\epsilon)$ is

$$O\left(\frac{n^\omega}{\epsilon}\log^2\left(\frac{n^2W}{\epsilon}\right)\log\left(\frac{n^2W^2}{\epsilon}\right)\log^2\left(\frac{\log\left(\frac{n^2W}{\epsilon}\right)}{\epsilon}\right)\log(n)\right).$$

With $\log(n^2W) \le \log((nW)^2) = O(\log(nW))$ we get that Algorithm 1 runs in time

$$O\left(\frac{n^\omega}{\epsilon}\log^3\left(\frac{nW}{\epsilon}\right)\log^2\left(\frac{\log\left(\frac{nW}{\epsilon}\right)}{\epsilon}\right)\log(n)\right). \tag{10}$$

□

Note that since weights are integral and by Corollary 3 there is a simple cycle with minimum mean weight, the distance between two possible values for the minimum cycle mean is $\ge 1/n(n-1)$ [11]. Thus the minimum cycle mean could be determined exactly from $\hat{\mu}$ with $\epsilon \le 1/(n^2W)$. However, this would give a running time that is worse than known exact algorithms.

5. Finding an approximately optimal cycle

In this section we provide an algorithm that outputs, for $\mu > 0$, a cycle with mean weight of at most $(1+\epsilon)\mu$ for ϵ chosen as in Theorem 13. The algorithm uses the intermediate results of Algorithm 1, i.e., the matrices F_s for $1 \le s \le \log t$ which approximate D^{2^s}. For $t = O(n^2W/\epsilon)$ it needs $O(n^2\log(nW/\epsilon))$ space compared to $O(n^2)$ for computing only the approximate minimum cycle mean. The running time of the algorithm is $O(n^2\log(nW/\epsilon)/\epsilon)$ plus the running time of Algorithm 1, where the former is dominated by the running time of Algorithm 1.

The main idea of our algorithm for cycle extraction is as follows. Consider a path P of length t starting at x with approximately minimum weight. Let y be the final vertex of this path. This path can be described as the sum of a simple path from x to y and a set of cycles. If we subtract the simple path, the mean weight of the remaining part might become larger, but not too large because a simple path can have at most n edges, while the whole path has $t = O(n^2W/\epsilon)$ edges. Given a set of cycles and an upper bound on the mean weight of the edges in these cycles, there must be a simple cycle with mean at most this upper bound. If we can efficiently find this cycle, we can output it as a cycle with approximately minimum mean weight. However, spending time proportional to the path length t would already be too costly. Therefore, we partition the path into $O(n/\epsilon)$ non-overlapping *segments* of equal size and consider the corresponding path P' in a graph G' where the only edges are these segments. In G' there are $O(n/\epsilon)$ edges, thus we can find a cycle C in G' with minimum mean weight and at most n edges in $O(n^2/\epsilon)$ time with Karp's algorithm. We can split each edge of G', which corresponds to a segment of the original path P, in half by obtaining the *midpoint* of the segment, i.e., the vertex in the middle of the corresponding path segment. Let G'' be the graph consisting of the segment halves of the edges in the cycle C. Since the cycle C has at most n edges, the graph G'' has at most $2n$ edges. Thus we can run Karp's algorithm on G'' in $O(n^2)$ time. We repeat the halving of the segments until the remaining segments correspond to edges in the original graph and we can indeed output a cycle with approximately minimum mean weight.

In the implementation of the algorithm we will need to obtain the midpoint of a path segment. A segment will correspond to an entry $F_s[u,v]$ in the matrix that approximates D^{2^s} for some segment length $l = 2^s$ with $1 \leq s \leq \log t$. Thus we can obtain a midpoint z such that $F_{s-1}[u,z] + F_{s-1}[z,v]$ is approximately equal to $d_l(u,v)$ in time $O(n)$ by computing $\arg\min_{z' \in V}(F_{s-1}[u,z'] + F_{s-1}[z',v])$.

We will consider a graph whose edges are exactly the edges of a (non-simple) path. To show that in this graph there exists a simple cycle with mean weight at most the mean weight of all the edges in the path, we will view the graph as a multigraph and apply Propositions 1 and 2. Note that parts of the path might be traversed multiple times. Thus we need to consider the multiset of edges in the path in order to argue about the ratio of the sum of the weights to the number of edges. The multigraph is only needed in the analysis, not in the actual algorithm.

Algorithm 2: Cycle with approximately minimum mean weight

```
input  : $F_s$ for $s = 1, \ldots, \log t$ with $D^{2^s}[i,j] \leq F_s[i,j] \leq \left(1 + \frac{4}{R}\right)^s D^{2^s}[i,j]$ and $\left(1 + \frac{4}{R}\right)^{\log t} \leq (1 + \epsilon)$
output : cycle with mean weight $\leq (1 + \epsilon) \min_{(i,j)} F_{\log t}[i,j]/t$
1  $(x,y) \leftarrow \arg\min_{(i,j)}(F_{\log t}[i,j])$ and let $P$ denote the corresponding path
2  $L \leftarrow \epsilon t/(2n)$
3  $l \leftarrow 2^{\lfloor \log L \rfloor}$
4  $E' \leftarrow \{(x,y)\}$
5  for $j \leftarrow 1$ to $t/l$ do                     /* split $P$ into $O(n/\epsilon)$ segments */
6      $E'' \leftarrow \emptyset$
7      foreach $(u,v) \in E'$ do
8          find midpoint $z$ of segment corresponding to $(u,v)$
9          $E'' \leftarrow E'' \cup \{(u,z),(z,v)\}$
10     end
11     $E' \leftarrow E''$
12 end
13 for $\forall u,v \in V$ and $s = 1, \ldots, \log t$ let $w_{2^s}(u,v) := F_s[u,v]$     /* weight function */
14 $G' \leftarrow (V, E', w_l)$
15 find cycle $C$ with minimum mean weight and $\leq n$ edges in $G'$
16 while $l > 1$ do                                   /* cycle refinement */
17     $l \leftarrow l/2$
18     $E' \leftarrow \emptyset$
19     foreach $(u,v) \in C$ do
20         find midpoint $z$ of segment corresponding to $(u,v)$
21         $E' \leftarrow E' \cup \{(u,z),(z,v)\}$
22     end
23     $G' \leftarrow (V, E', w_l)$
24     find cycle $C$ with minimum mean weight and $\leq n$ edges in $G'$
25 end
26 return $C$
```

Lemma 14. *Given all intermediate results of Algorithm 1 for some $0 < \epsilon \le 1$ and a power of two $t \ge 1$, Algorithm 2 outputs a cycle with mean weight $\hat{\mu}$ such that $\hat{\mu} \le (1+\epsilon)\min_{x \in V} \hat{\delta}_t(x)/t$ in time $O((n^2/\epsilon)\log t)$.*

Proof. Let F_s denote the intermediate results of Algorithm 1 for $1 \le s \le \log t$. Further let $(x, y) = \arg\min_{(i,j)}(F_{\log t}[i, j])$, i.e., $x = \arg\min_{x' \in V} \hat{\delta}_t(x')$. Let P be a path from x to y with length t and weight $\delta_t(x) \le w(P) \le \hat{\delta}_t(x)$ such that $d_t(x, y) \le \hat{\delta}_t(x) \le (1+\epsilon)d_t(x, y)$. Remember that by Equation (1) each entry $F_s[i, j]$ is a $(1+4/R)^s$-approximation of $d_{2^s}(i, j)$ for all vertices i, j and $1 \le s \le \log t$ and that by Equation (2) we have $(1+4/R)^{\log t} \le 1+\epsilon$. Thus, for any length $l \le t$ that is a power of 2 and any pair of vertices (u, v) with $d_l(u, v)$, we can obtain a midpoint z such that $d_l(u, v) \le F_{\log(l/2)}[u, z] + F_{\log(l/2)}[z, v] \le (1+4/R)^{\log l} d_l(u, v)$ in time $O(n)$ by computing $\arg\min_{z' \in V}(F_{\log(l/2)}[u, z'] + F_{\log(l/2)}[z', v])$.

In Algorithm 2 (lines 5–12) we divide the path P into $O(n/\epsilon)$ segments of equal size l such that l is the largest power of 2 smaller or equal $\epsilon t/(2n)$. We do this by repeatedly doubling the number of segments we split the path P into, starting with the segment (x, y), which represents the whole path P. In each iteration a midpoint for each of the current segments is found, thus in total less than $\sum_{i=0}^{\log(4n/\epsilon)} 2^i = O(n/\epsilon)$ midpoints have to be found. Hence splitting path P into $O(n/\epsilon)$ segments takes time $O(n^2/\epsilon)$.

Let E_l be the set of all segments of P and let G_l be the graph with vertices V and edges E_l where the edges have weight $w_l(i, j) = F_{\log l}[i, j]$. We will prove below that there exists a simple cycle in G_l with mean weight at most $\hat{\mu} \le (1+\epsilon)\hat{\delta}_t(x)/t$. Given the existence of such a cycle, we can find a cycle C in G_l with at most n segments and minimum mean weight $\le \hat{\mu}$ with Karp's algorithm [2] in $O(n^2/\epsilon)$ time (line 15 of Algorithm 2) because the number of edges in G_l is $O(n/\epsilon)$.

To show that there exists a simple cycle in G_l with mean weight at most $\hat{\mu} \le (1+\epsilon)\hat{\delta}_t(x)/t$, we consider the multigraph G'_l with vertices V and edges E'_l with weight $w_l(i, j) = F_{\log l}[i, j]$, where E'_l is the multiset of all segments of P. Note that by Theorem 7 the mean weight of the edges in G'_l is bounded above by $\hat{\delta}_t(x)/t$. Since $|E'_l| \ge n$, we have that G'_l contains at least one cycle. By Proposition 1, the set E'_l can be partitioned into a simple path from x to y and a set of simple cycles S. The simple path from x to y can contain at most n segments, where each segment represents $l \le \epsilon t/(2n)$ edges in the original graph. Since we assume nonnegative edge weights, the mean weight $\hat{\mu}$ of the segments in S is at most

$$\hat{\mu} \le \frac{\hat{\delta}_t(x)}{t - n\frac{\epsilon t}{2n}} \le \frac{1}{1-\frac{\epsilon}{2}} \frac{\hat{\delta}_t(x)}{t} \le (1+\epsilon)\frac{\hat{\delta}_t(x)}{t} \ .$$

By Proposition 2 there exists a simple cycle with mean weight at most $\hat{\mu} \le (1+\epsilon)\hat{\delta}_t(x)/t$ in S and in the multigraph G'_l, and thus in the simple graph G_l.

As explained above, we can find the midpoint z of each segment $(u, v) \in C$ in $O(n)$ time. Thus we can obtain a multigraph $G'_{l/2}$ with vertices V, the two edges (u, z) and (z, v) for each $(u, v) \in C$, and weights $w_{l/2}(i, j) = F_{\log(l/2)}[i, j]$ in time $O(n^2)$. By Proposition 1 the edges in $G'_{l/2}$ can be described as a set of simple cycles. By Proposition 2 there exists a simple cycle C' in $G'_{l/2}$ with mean weight at most $\hat{\mu}$. Given the existence, we again only have to consider the corresponding simple graph $G_{l/2}$, for which we can obtain a minimum mean cycle with at most n edges with Karp's algorithm in time $O(n^2)$ (line 24 of Algorithm 2). We can repeat this process of reducing the segment size for the found simple cycle until each segment corresponds to an edge in the original graph and we thus have indeed found a cycle with mean weight at most $\hat{\mu}$.

Since we halve the segment length in each iteration, at most $\log l$ iterations are needed. Hence the running time of the while-loop (lines 16–25 of Algorithm 2) can be bounded by $O(n^2 \log t)$, which implies a total running time of $O((n^2/\epsilon)\log t)$. □

Remark 15. *In Algorithm 2 above we use the naive $O(n)$-time procedure for finding midpoints. A more efficient approach is the following: We modify the procedure* approx-min-plus *in Algorithm 1 to additionally output a matrix of* witnesses *as described in [34]. For $C := F_s \star F_s$ the matrix of witnesses contains for each entry (i, j) in F_{s+1} an index $1 \le k \le n$ such that $C[i, j] \le F_s[i, k] + F_s[k, j] \le (1+4/R)C[i, j]$. Having stored the witness matrices, a midpoint (witness) can be found in constant time.*

Theorem 16. *Given a graph with nonnegative integer edge weights, we can compute a cycle with mean weight at most $\hat{\mu}$ such that $\mu \leq \hat{\mu} \leq (1+\epsilon)\mu$ for $0 < \epsilon \leq 1$ in*

$$\widetilde{O}\left(\frac{n^\omega}{\epsilon}\log^3\left(\frac{nW}{\epsilon}\right)\right) \text{ time and } O\left(n^2\log\left(\frac{nW}{\epsilon}\right)\right) \text{ space.}$$

Proof. If $\mu = \min_{x' \in V} \mu(x') = 0$, we can find a cycle with mean weight of zero in $O(n^2)$ time by Proposition 12. If $\mu > 0$, we set $\epsilon' = \epsilon/7$ like in the proof of Theorem 13. We execute Algorithm 1 with weight matrix D, error bound ϵ' and t such that t is the smallest power of two with $t \geq n^2 W/\epsilon'$ and save all intermediate results F_s for $s = 1, \ldots, \log t$. This gives the claimed running time by Theorem 13 and requires $O(n^2 \log t)$ space.

By Lemma 14 the running time to output a cycle given the intermediate results of Algorithm 1 is subsumed by the running time of Algorithm 1. It remains to show the approximation guarantee on the mean weight of the cycle computed by Algorithm 2. The lower bound is given trivially since we actually output a cycle in the original graph. Let $x = \arg\min_{x' \in V} \mu(x')$. By Lemma 10 and the choice of t (Equation (9)) we have $\delta_t(x)/t \leq (1+\epsilon')\mu(x)$. Lemma 9 gives $\hat{\delta}_t(x) \leq (1+\epsilon')\delta_t(x)$. Combined with Lemma 14 this yields an upper bound on the error of the mean weight $\hat{\mu}$ of the output cycle of

$$\hat{\mu} \leq (1+\epsilon')^3\mu \leq (1+7\epsilon')\mu \leq (1+\epsilon)\mu \,. \qquad \square$$

6. Conclusion

We hope that this work draws attention to the problem of approximating the minimum cycle mean. It would be interesting to study whether there is a faster approximation algorithm for the minimum cycle mean problem, maybe at the cost of a worse approximation. The running time of our algorithm immediately improves if faster algorithms for classic matrix multiplication, min-plus matrix multiplication or approximate min-plus multiplication are found. However, a different approach might lead to better results and might shed new light on how well the problem can be approximated. Therefore it would be interesting to remove the dependence on fast matrix multiplication and develop a so-called combinatorial algorithm.

Acknowledgements

K. C. is supported by the Austrian Science Fund (FWF): P23499-N23 and S11407-N23 (RiSE), an ERC Start Grant (279307: Graph Games), and a Microsoft Faculty Fellows Award. M. H. is supported by the Austrian Science Fund (FWF): P23499-N23, the Vienna Science and Technology Fund (WWTF) grant ICT10-002, and the University of Vienna (IK I049-N). S. K. is supported by the Austrian Science Fund (FWF): P23499-N23 and the University of Vienna (IK I049-N). V. L. is supported by the Vienna Science and Technology Fund (WWTF) grant ICT10-002. M. R. is supported by the Russian Foundation for Basic Research (RFBR) research project No. 12-01-00864-a.

References

[1] K. Chatterjee, M. Henzinger, S. Krinninger, V. Loitzenbauer, Approximating the minimum cycle mean, in: GandALF, 2013, pp. 136–149. doi:10.4204/EPTCS.119.13.

[2] R. M. Karp, A characterization of the minimum cycle mean in a digraph, Discrete Mathematics 23 (3) (1978) 309–311. doi:10.1016/0012-365X(78)90011-0.

[3] J. B. Orlin, R. K. Ahuja, New scaling algorithms for the assignment and minimum mean cycle problems, Mathematical Programming 54 (1-3) (1992) 41–56. doi:10.1007/BF01586040.

[4] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, Network flows: theory, algorithms, and applications, Prentice Hall, 1993.

[5] R. Bloem, K. Greimel, T. A. Henzinger, B. Jobstmann, Synthesizing robust systems, in: FMCAD, 2009, pp. 85–92. doi:10.1109/FMCAD.2009.5351139.

[6] M. Droste, I. Meinecke, Describing Average- and Longtime-Behavior by Weighted MSO Logics, in: MFCS, 2010, pp. 537–548. doi:10.1007/978-3-642-15155-2_47.

[7] V. Vassilevska Williams, Multiplying Matrices Faster Than Coppersmith-Winograd, in: STOC, 2012, pp. 887–898. doi:10.1145/2213977.2214056.

[8] F. Le Gall, Powers of Tensors and Fast Matrix Multiplication, in: ISSAC, 2014, p. 23. doi:10.1145/2608628.2627493.

[9] P. Sankowski, Shortest Paths in Matrix Multiplication Time, in: ESA, 2005, pp. 770–778. doi:10.1007/11561071_68.

[10] P. Butkovic, R. A. Cuninghame-Green, An $O(n^2)$ algorithm for the maximum cycle mean of an $n \times n$ bivalent matrix., Discrete Applied Mathematics 35 (2) (1992) 157–162. doi:10.1016/0166-218X(92)90039-D.

[11] U. Zwick, M. Paterson, The complexity of mean payoff games on graphs, Theoretical Computer Science 158 (1-2) (1996) 343–359, announced at COCOON '95. doi:10.1016/0304-3975(95)00188-3.

[12] A. Ehrenfeucht, J. Mycielski, Positional strategies for mean payoff games, International Journal of Game Theory 8 (2) (1979) 109–113. doi:10.1007/BF01768705.

[13] V. Gurvich, A. Karzanov, L. Khachiyan, Cyclic games and an algorithm to find minimax cycle means in directed graphs, USSR Computational Mathematics and Mathematical Physics 28 (5) (1988) 85–91. doi:10.1016/0041-5553(88)90012-2.

[14] M. Hartmann, J. B. Orlin, Finding Minimum Cost to Time Ratio Cycles With Small Integral Transit Times, Networks 23 (6) (1993) 567–574. doi:10.1002/net.3230230607.

[15] A. Dasdan, R. K. Gupta, Faster Maximum and Minimum Mean Cycle Algorithms for System-Performance Analysis, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 17 (10) (1998) 889–899. doi:10.1109/43.728912.

[16] R. M. Karp, J. B. Orlin, Parametric shortest path algorithms with an application to cyclic staffing, Discrete Applied Mathematics 3 (1) (1981) 37–45. doi:10.1016/0166-218X(81)90026-3.

[17] N. E. Young, R. E. Tarjan, J. B. Orlin, Faster Parametric Shortest Path and Minimum-Balance algorithms, Networks 21 (2) (1991) 205–221. doi:10.1002/net.3230210206.

[18] O. Madani, Polynomial Value Iteration Algorithms for Deterministic MDPs, in: UAI, 2002, pp. 311–318.

[19] R. A. Howard, Dynamic Programming and Markov Processes, MIT Press, 1960.

[20] T. D. Hansen, U. Zwick, Lower Bounds for Howard's Algorithm for Finding Minimum Mean-Cost Cycles, in: ISAAC, 2010, pp. 415–426. doi:10.1007/978-3-642-17517-6_37.

[21] P. B. Miltersen, Recent Results on Howard's Algorithm, in: MEMICS, 2013, pp. 53–56. doi:10.1007/978-3-642-36046-6_6.

[22] E. L. Lawler, Combinatorial optimization: Networks and Matroids, Dover Publications, 1976.

[23] E. Zemel, A Linear Time Randomizing Algorithm for Searching Ranked Functions, Algorithmica 2 (1-4) (1987) 81–90. doi:10.1007/BF01840350.

[24] A. V. Goldberg, Scaling Algorithms for the Shortest Paths Problem, SIAM Journal on Computing 24 (3) (1995) 494–504. doi:10.1137/S0097539792231179.

[25] S. T. McCormick, Approximate Binary Search Algorithms for Mean Cuts and Cycles, Operations Research Letters 14 (3) (1993) 129–132. doi:10.1016/0167-6377(93)90022-9.

[26] A. Roth, M.-F. Balcan, A. Kalai, Y. Mansour, On the Equilibria of Alternating Move Games, in: SODA, 2010, pp. 805–816.

[27] E. Boros, K. Elbassioni, M. Fouz, V. Gurvich, K. Makino, B. Manthey, Stochastic Mean Payoff Games: Smoothed Analysis and Approximation Schemes, in: ICALP, 2011, pp. 147–158. doi:10.1007/978-3-642-22006-7_13.

[28] V. Vassilevska Williams, R. Williams, Subcubic Equivalences between Path, Matrix and Triangle Problems, in: FOCS, 2010, pp. 645–654. doi:10.1109/FOCS.2010.67.

[29] M. L. Fredman, New Bounds on the Complexity of the Shortest Path Problem, SIAM Journal on Computing 5 (1) (1976) 83–89. doi:10.1137/0205006.

[30] T. M. Chan, More Algorithms for All-Pairs Shortest Paths in Weighted Graphs, SIAM Journal on Computing 39 (5) (2010) 2075–2089, announced at STOC '07. doi:10.1137/08071990X.

[31] R. Williams, Faster all-pairs shortest paths via circuit complexity, in: STOC, 2014, pp. 664–673. doi:10.1145/2591796.2591811.

[32] G. Yuval, An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications, Information Processing Letters 4 (6) (1976) 155–156. doi:10.1016/0020-0190(76)90085-5.

[33] N. Alon, Z. Galil, O. Margalit, On the Exponent of the All Pairs Shortest Path Problem, Journal of Computer and System Sciences 54 (2) (1997) 255–262, announced at FOCS '91. doi:10.1006/jcss.1997.1388.

[34] U. Zwick, All Pairs Shortest Paths using Bridging Sets and Rectangular Matrix Multiplication, Journal of the ACM 49 (3) (2002) 289–317, announced at FOCS '98. doi:10.1145/567112.567114.

[35] C.-H. Wong, Y.-C. Tam, Negative Cycle Detection Problem, in: ESA, 2005, pp. 652–663. doi:10.1007/11561071_58.

[36] B. V. Cherkassky, A. V. Goldberg, Negative-cycle detection algorithms, Mathematical Programming 85 (2) (1999) 277–311, announced at ESA '96. doi:10.1007/s101070050058.

[37] R. Gentilini, A note on the approximation of mean-payoff games, in: CILC, 2011, pp. 333–340.

[38] A. V. Aho, J. E. Hopcroft, J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.