Approximating the minimum cycle mean

Krishnendu Chatterjee*
IST Austria
Institute of Science and Technology
Klosterneuburg, Austria

Monika Henzinger†, Sebastian Krinninger‡, Veronika Loitzenbauer§
University of Vienna
Faculty of Computer Science
Vienna, Austria

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 in $O(n^2)$ time 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 $(1+\varepsilon)$-approximation algorithm for the problem and the running time of our algorithm is $\widetilde{O}(n^\omega \log^3(nW/\varepsilon)/\varepsilon)$[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.

1 Introduction

Minimum cycle mean problem. We consider a fundamental graph 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, 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 [17]; and the current best known algorithm for the problem, which is over two decades old, by Orlin and Ahuja require $O(m\sqrt{n}\log(nW))$ time [22], 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 [2]. 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 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

*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.

†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).

‡Supported by the Austrian Science Fund (FWF): P23499-N23 and the University of Vienna (IK I049-N).

§Supported by the Vienna Science and Technology Fund (WWTF) grant ICT10-002.

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

Gabriele Puppis, Tiziano Villa (Eds.): Fourth International Symposium on Games, Automata, Logics and Formal Verification
EPTCS 119, 2013, pp. 136–149, doi:10.4204/EPTCS.119.13

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 objectives*) [4, 9].

Results. This work contains the following results.

1. *Reduction to min-plus matrix multiplication.* We show that the minimum cycle mean problem is reducible in $O(n^2)$ time 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 and $O(n^2)$ additive factor in the running time.
2. *Faster approximation algorithm.* When the weights are nonnegative, we present the first $(1+\varepsilon)$-approximation algorithm for the problem that outputs $\hat{\mu}$ such that $\mu \leq \hat{\mu} \leq (1+\varepsilon)\mu$ and the running time of our algorithm is $\widetilde{O}(n^\omega \log^3(nW/\varepsilon)/\varepsilon)$. 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 $\omega < 2.3727$. The worst case complexity of the current best known algorithm for the minimum cycle mean problem is $O(m\sqrt{n}\log(nW))$ [22], which could be as bad as $O(n^{2.5}\log(nW))$. Thus for $(1+\varepsilon)$-approximation our algorithm provides better dependence in 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 $(1+\varepsilon)$-approximation of the minimum cycle mean problem with small weights and large graphs is a very 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 [17]	$O(mn)$	exact	$[-W,W]$
Orlin and Ahuja [22]	$O(m\sqrt{n}\log(nW))$	exact	$[-W,W]\cap\mathbb{Z}$
Sankowski [24] (implicit)	$\widetilde{O}(Wn^\omega\log(nW))$	exact	$[-W,W]\cap\mathbb{Z}$
Butkovic and Cuninghame-Green [6]	$O(n^2)$	exact	$\{0,1\}$
This paper	$\widetilde{O}(n^\omega\log^3(nW/\varepsilon)/\varepsilon)$	$1+\varepsilon$	$[0,W]\cap\mathbb{Z}$

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

1.1 Related work

The minimum cycle mean problem is basically equivalent to solving a deterministic Markov decision process (MDP) [31]. The latter can also be seen as a single-player mean-payoff game [10, 13, 31]. 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 [17] uses a dynamic programming approach to find the minimum cycle mean and runs in time $O(mn)$. 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 [15] and of Dasdan and Gupta [8] 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 [18] can compute the minimum cycle mean in time $O(mn\log n)$. Young, Tarjan, and Orlin [27] 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 [20] 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 [16]. The complexity of this algorithm for deterministic MDPs is unresolved. Recently, Hansen and Zwick [14] provided a class of weighted graphs on which Howard's algorithm performs $\Omega(n^2)$ iterations where each iteration takes time $O(m)$.

Algorithms depending on weights. If a graph is complete and has only two different edge weights, then the minimum cycle mean problem problem can be solved in time $O(n^2)$ because the matrix of its weights is bivalent [6].

Another approach is to use the connection to the problem of detecting a negative cycle. Lawler [19] 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 [22] extend this idea by the approximate binary search technique [29]. By combining approximate binary search with their scaling algorithm for the assignment problem they can compute the minimum cycle mean 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.[2] 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 [12] solves the SSSP problem (and therefore also the negative cycle detection problem) in time $O(m\sqrt{n}\log W)$. McCormick [21] 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 [22]. Sankowski's matrix multiplication based algorithm [24] 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 [23, 5], 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 [23] and [5] are obtained by reductions to a pseudo-polynomial algorithm for solving mean-payoff games. In the case of the minimum cycle mean problem, these reductions do not provide an improvement over the current fastest exact algorithms mentioned above.

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

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 $\omega < 2.3727$ [25]. Moreover, Williams and Williams [26] showed that computing the min-plus product is computationally equivalent to a series of problems including all-pairs shortest paths and negative triangle detection. 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 by logarithmic factors and by assuming small integer entries.

Fredman [11] gave an algorithm for computing the min-plus product with a slightly subcubic running time of $O(n^3(\log\log n)^{1/3}/(\log n)^{1/3})$. This algorithm is "purely combinatorial", i.e., it does not rely on fast algorithms for classic matrix multiplication. After a long line of improvements, the current fastest such algorithm by Chan [7] runs in time $O(n^3(\log\log n)^3/(\log n)^2)$.

A different approach for computing the min-plus product of two integer matrix is to reduce the problem to classic matrix multiplication [28]. 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 [3]. This observation was used by Alon, Galil, and Margalit [3] and Zwick [30] 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 $(1+\varepsilon)$-approximate all-pairs shortest paths in graphs with non-negative edge weights. Our approach of finding the minimum cycle mean extensively uses this technique.

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. A *cycle* is a path in which the start vertex and the end vertex are the same. The *length of a path* P is the number of edges of P. 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 *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. 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 $A \otimes B$. The min-plus product is defined as follows. If $C = A \otimes B$, 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 $A^{t+1} = A \otimes A^t$ 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 $\omega < 2.3727$ [25].

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 ($t = O(n^3 W)$) [31].[3] 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 1. *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 $D \otimes D^{t-1} = D^t$ 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 2. *For every $t \geq 1$ we have $D^{2t} = D^t \otimes D^t$. Therefore the matrix D^t can be computed with $O(\log t)$ many min-plus matrix multiplications.*

[3] Specifically, for $t = 4n^3W$ 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 $\mu(x)$ [31].

Proof. We give the proof for the sake of completeness. It can easily be verified that the min-plus matrix product is associative [1] and therefore $D^{2t} = D^t \otimes D^t$. 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 $D^t = D^{t_1} \otimes \ldots \otimes D^{t_k}$ 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 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 $O(\log t)$. □

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.

Theorem 3. *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 $t = O(n^3 W)$.*[4]

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 $t = O(n^2 W/\varepsilon)$), 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 4 (Zwick [30])**.** *Let A and B be two $n \times n$ matrices with integer entries in $[0, M]$ and let $C := A \otimes B$. Let $R \geq \log n$ be a power of two. The algorithm* approx-min-plus(A, B, M, R) *computes the*

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

approximate min-plus product $\overline{C}$ *of* A *and* B *in time*[5] $O(n^\omega R\log(M)\log^2(R)\log(n))$ *such that for every* $1 \le i,j \le n$ *it holds that* $C[i,j] \le \overline{C}[i,j] \le (1+4/R)C[i,j]$.

We now give a modification (see Algorithm 1) of Zwick's algorithm for approximate shortest paths [30] such that the algorithm computes a $(1+\varepsilon)$-approximation F of D^t when t is a power of two such that for $1 \le i,j \le n$ we have $D^t[i,j] \le F[i,j] \le (1+\varepsilon)D^t[i,j]$. Just as we can compute D^t exactly with $\log t$ min-plus matrix multiplications, the algorithm computes the $(1+\varepsilon)$-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 ε, t (a power of 2)
output : $(1+\varepsilon)$-approximation of D^t

$F \leftarrow D$
$r \leftarrow 4\log t/\ln(1+\varepsilon)$
$R \leftarrow 2^{\lceil \log r \rceil}$
for $\log t$ *times* **do**
  $F \leftarrow$ approx-min-plus$(F, F, 2tW, R)$
end
return F

Lemma 5. *Given an* $0 < \varepsilon \le 1$ *and a power of two* $t \ge 1$, *Algorithm 1 computes a* $(1+\varepsilon)$*-approximation* F *of* D^t *in time*

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

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

Proof. The basic idea is as follows. 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 $4\log(t)/\ln(1+\varepsilon)$ suffices to bound the approximation error by $(1+\varepsilon)$. We will show that $2tW$ is an upper bound on the entries in the input matrices for approx-min-plus. The stated running time follows directly from these two facts and Theorem 4.

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

[5]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 4 if $R \ge \log n$, which will always be the case in the following.

This results in an overall running time of

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

The last equation follows from the inequality $1/\ln(1+\varepsilon) \leq (1+\varepsilon)/\varepsilon$ for $\varepsilon > 0$. Since $\varepsilon \leq 1$ it follows that $1/\log(1+\varepsilon) = O(1/\varepsilon)$.

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

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

holds after the s-th iteration of Algorithm 1 by induction on s. Note that the $(1+\varepsilon)$-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

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

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., $C_{s+1} = F_s \otimes F_s$. 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

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

By the induction hypothesis and the definition of k_c we have

$$D_s[i,k_c] + D_s[k_c,j] \leq F_s[i,k_c] + F_s[k_c,j] = C_{s+1}[i,j]. \tag{2}$$

By Theorem 4 the values of F_{s+1} can only be larger than the values in C_{s+1}, i.e.,

$$C_{s+1}[i,j] \leq F_{s+1}[i,j]. \tag{3}$$

Combining Equations (1), (2), and (3) 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 4 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

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

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

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

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:

$$\begin{aligned} F_s[i,k_d] + F_s[k_d,j] &\leq \left(1+\frac{4}{R}\right)^s D_s[i,k_d] + \left(1+\frac{4}{R}\right)^s D_s[k_d,j]\,, \\ &= \left(1+\frac{4}{R}\right)^s \left(D_s[i,k_d] + D_s[k_d,j]\right)\,, \\ &= \left(1+\frac{4}{R}\right)^s D_{s+1}[i,j]\,. \end{aligned} \tag{6}$$

Combining Equations (4), (5), and (6) 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 1. 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 6. *The value* $\hat{\delta}_t(x) := \min_{y\in V} F[x,y]$ *approximates* $\delta_t(x)$ *with* $\delta_t(x) \leq \hat{\delta}_t(x) \leq (1+\varepsilon)\delta_t(x)$.

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.,

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

By these definitions and Lemma 5 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

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

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 [31] 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.

Lemma 7. *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 weight 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. 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 weight 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

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

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 $(1+\varepsilon)$-approximation $\hat{\mu}(x)$ of $\mu(x)$ even if we only have an approximation $\hat{\delta}_t(x)$ of $\delta_t(x)$ with $(1+\varepsilon)$-error. We exclude the case $\mu(x) = 0$ for the moment.

Lemma 8. *Assume we have an approximation $\hat{\delta}_t(x)$ of $\delta_t(x)$ such that $\delta_t(x) \leq \hat{\delta}_t(x) \leq (1+\varepsilon)\delta_t(x)$ for $0 < \varepsilon \leq 1/2$. If*

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

then

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

Proof. We first show that $\hat{\mu}(x)$ is at least as large as $\mu(x)$. From Lemma 7 we have $\delta_t(x) \geq t \cdot \mu(x) - nW$. As t is chosen large enough,

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

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

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

For the upper bound on $\hat{\mu}(x)$ we use the inequality $\delta_t(x) \leq t \cdot \mu(x) + nW$ from Lemma 7. As t is chosen large enough,

$$\frac{\delta_t(x)}{t} \leq \mu(x) + \frac{nW}{t} \leq \mu(x) + \frac{\varepsilon}{n} \leq (1+\varepsilon)\mu(x).$$

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

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

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

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

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 9. *Given a graph with nonnegative integer edge weights, we can find out all vertices x such that $\mu(x) = 0$ 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. First, we compute the strongly connected components of G, the original graph. Each strongly connected component G_i (where $1 \leq i \leq k$) is a subgraph of G with a set of vertices V_i and a set of edges E_i. For every $1 \leq i \leq k$ we let $G_i^0 = (E_i^0, V_i)$ denote the subgraph of G_i that only contains edges of weight 0, i.e., $E_i^0 = \{e \in E_i | w(e) = 0\}$. As argued above, G_i contains a zero-mean cycle if and only if G_i^0 contains a cycle. We can check whether G_i^0 contains a cycle by computing the strongly connected components of G_i^0: G_i^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 strongly connected components of G that contain a zero-mean cycle. The vertices in Z are not the only vertices that can reach a zero-mean cycle. 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.

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 8 tells us that $t = n^2W/\varepsilon$ 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 10. *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 $\mu(x) \leq \hat{\mu}(x) \leq (1+\varepsilon)\mu(x)$ for $0 < \varepsilon \leq 1$ in time*

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

Proof. First we find all vertices x with $\mu(x) = 0$. By Proposition 9 this takes time $O(n^2)$ for $m = O(n^2)$. For the remaining vertices x we approximate $\mu(x)$ as follows.

Let $\varepsilon' := \varepsilon/7$. If we execute Algorithm 1 with weight matrix D, error bound ε' and t such that t is the smallest power of two with $t \geq n^2 W/\varepsilon'$, we obtain a $(1+\varepsilon')$-approximation $F[x,y]$ of $D^t[x,y]$ for all vertices x and y (Lemma 5). By calculating for every x the minimum entry of $F[x,y]$ over all y we have a $(1+\varepsilon')$-approximation of $\delta_t(x)$ (Lemma 6). By Lemma 8 $\hat{\mu}(x) := \hat{\delta}_t(x)/((1-\varepsilon')t)$ is for this choice of t an approximation of $\mu(x)$ such that $\mu(x) \leq \hat{\mu}(x) \leq (1+7\varepsilon')\mu(x)$. By substituting ε' with $\varepsilon/7$ we get $\mu(x) \leq \hat{\mu}(x) \leq (1+\varepsilon)\mu(x)$ i.e., a $(1+\varepsilon)$-approximation of $\mu(x)$.

By Lemma 5 the running time of Algorithm 1 for $t = 2^{\lceil \log(n^2 W/\varepsilon') \rceil} = O(n^2 W/\varepsilon)$ is

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

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

$$O\left(\frac{n^\omega}{\varepsilon} \log^3\left(\frac{nW}{\varepsilon}\right) \log^2\left(\frac{\log\left(\frac{nW}{\varepsilon}\right)}{\varepsilon}\right) \log(n)\right). \tag{7}$$

□

5 Open problems

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.

Another obvious extension is to allow negative edge weights in the input graph. Furthermore, we only consider the minimum cycle mean problem, while it might be interesting to actually output a cycle with approximately optimal mean weight.

References

[1] Alfred V. Aho, John E. Hopcroft & Jeffrey D. Ullman (1974): *The Design and Analysis of Computer Algorithms*. Addison-Wesley.

[2] Ravindra K. Ahuja, Thomas L. Magnanti & James B. Orlin (1993): *Network flows: theory, algorithms, and applications*. Prentice Hall.

[3] Noga Alon, Zvi Galil & Oded Margalit (1997): *On the Exponent of the All Pairs Shortest Path Problem*. *Journal of Computer and System Sciences* 54(2), pp. 255–262, doi:10.1006/jcss.1997.1388. Announced at FOCS '91.

[4] Roderick Bloem, Karin Greimel, Thomas A. Henzinger & Barbara Jobstmann (2009): *Synthesizing robust systems*. In: *FMCAD*, pp. 85–92, doi:10.1109/FMCAD.2009.5351139.

[5] Endre Boros, Khaled Elbassioni, Mahmoud Fouz, Vladimir Gurvich, Kazuhisa Makino & Bodo Manthey (2011): *Stochastic Mean Payoff Games: Smoothed Analysis and Approximation Schemes*. In: *ICALP*, pp. 147–158, doi:10.1007/978-3-642-22006-7_13.

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

[7] Timothy M. Chan (2010): *More Algorithms for All-Pairs Shortest Paths in Weighted Graphs*. *SIAM Journal on Computing* 39(5), pp. 2075–2089, doi:10.1137/08071990X. Announced at STOC '07.

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

[9] Manfred Droste & Ingmar Meinecke (2010): *Describing Average- and Longtime-Behavior by Weighted MSO Logics*. In: *MFCS*, pp. 537–548, doi:10.1007/978-3-642-15155-2_47.

[10] Andrzej Ehrenfeucht & Jan Mycielski (1979): *Positional strategies for mean payoff games*. *International Journal of Game Theory* 8(2), pp. 109–113, doi:10.1007/BF01768705.

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

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

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

[14] Thomas Dueholm Hansen & Uri Zwick (2010): *Lower Bounds for Howard's Algorithm for Finding Minimum Mean-Cost Cycles*. In: *ISAAC*, pp. 415–426, doi:10.1007/978-3-642-17517-6_37.

[15] Mark Hartmann & James B. Orlin (1993): *Finding Minimum Cost to Time Ratio Cycles With Small Integral Transit Times*. *Networks* 23(6), pp. 567–574, doi:10.1002/net.3230230607.

[16] Ronald A. Howard (1960): *Dynamic Programming and Markov Processes*. MIT Press.

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

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

[19] Eugène L. Lawler (1976): *Combinatorial optimization: Networks and Matroids*. Dover Publications.

[20] Omid Madani (2002): *Polynomial Value Iteration Algorithms for Deterministic MDPs*. In: *UAI*, pp. 311–318. Available at http://dl.acm.org/citation.cfm?id=2073913.

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

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

[23] Aaron Roth, Maria-Florina Balcan, Adam Kalai & Yishay Mansour (2010): *On the Equilibria of Alternating Move Games*. In: *SODA*, pp. 805–816. Available at http://dl.acm.org/citation.cfm?id=1873667.

[24] Piotr Sankowski (2005): *Shortest Paths in Matrix Multiplication Time*. In: *ESA*, pp. 770–778, doi:10.1007/11561071_68.

[25] Virginia Vassilevska Williams (2012): *Multiplying Matrices Faster Than Coppersmith-Winograd*. In: *STOC*, pp. 887–898, doi:10.1145/2213977.2214056.

[26] Virginia Vassilevska Williams & Ryan Williams (2010): *Subcubic Equivalences between Path, Matrix and Triangle Problems*. In: *FOCS*, pp. 645–654, doi:10.1109/FOCS.2010.67.

[27] Neal E. Young, Robert Endre Tarjan & James B. Orlin (1991): *Faster Parametric Shortest Path and Minimum-Balance algorithms*. *Networks* 21(2), pp. 205–221, doi:10.1002/net.3230210206.

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

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

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

[31] Uri Zwick & Mike Paterson (1996): *The complexity of mean payoff games on graphs*. *Theoretical Computer Science* 158(1-2), pp. 343–359, doi:10.1016/0304-3975(95)00188-3. Announced at COCOON '95.