Approximating the girth

Liam Roddity[*] Bar Ilan University

Roei Tov[†] Bar Ilan University

May 30, 2012

Abstract

This paper considers the problem of computing a minimum weight cycle in weighted undirected graphs. Given a weighted undirected graph $G = (V, E, w)$, let C be a minimum weight cycle of G, let $w(C)$ be the weight of C and let $w_{max}(C)$ be the weight of the maximal edge of C. We obtain three new approximation algorithms for the minimum weight cycle problem:

1. For integral weights from the range $[1, M]$ an algorithm that reports a cycle of weight at most $\frac{4}{3}w(C)$ in $O(n^2 \log n(\log n + \log M))$ time.
2. For integral weights from the range $[1, M]$ an algorithm that reports a cycle of weight at most $w(C) + w_{max}(C)$ in $O(n^2 \log n(\log n + \log M))$ time.
3. For non-negative real edge weights an algorithm that for any $\varepsilon > 0$ reports a cycle of weight at most $(\frac{4}{3} + \varepsilon)w(C)$ in $O(\frac{1}{\varepsilon}n^2 \log n(\log \log n))$ time.

In a recent breakthrough, Vassilevska Williams and Williams [WW10] showed that a subcubic algorithm, that computes the exact minimum weight cycle in undirected graphs with integral weights from the range $[1, M]$, implies a subcubic algorithm for computing all-pairs shortest paths in directed graphs with integral weights from the range $[-M, M]$. This implies that in order to get a subcubic algorithm for computing a minimum weight cycle, we have to relax the problem and to consider an approximated solution.

Lingas and Lundell [LL09] were the first to consider approximation in the context of minimum weight cycle in weighted graphs. They presented a 2-approximation algorithm for integral weights with $O(n^2 \log n(\log n + \log M))$ running time. They also posed, as an open problem, the question whether it is possible to obtain a subcubic algorithm with a c-approximation, where $c < 2$. The current paper answers this question in the affirmative, by presenting an algorithm with 4/3-approximation and the same running time.

Surprisingly, the approximation factor of 4/3 is not accidental. We show, using the new result of Vassilevska Williams and Williams [WW10], that a subcubic combinatorial algorithm with $(4/3 - \varepsilon)$-approximation, where $0 < \varepsilon \leq 1/3$, implies a subcubic combinatorial algorithm for multiplying two boolean matrices.

[*]`liamr@macs.biu.ac.il`. Work supported by the Israel Science Foundation (grant no. 822/10)

[†]`roei81@gmail.com`

1 Introduction

The problem of finding the girth (minimum weight cycle) of a graph is among the most basic and natural algorithmic graph problems. The problem in unweighted graphs was first considered at 1978 by Itai and Rodeh [IR78]. Among their results, they showed that a minimum weight cycle can be found by multiplying two boolean matrices. They also presented an $O(n^2)$ algorithm, that reports either a minimum cycle or a cycle with one edge more than the minimum cycle. These bounds remain the best known since then. Very recently, Vassilevska Williams and Williams [WW10] provided an explanation to the lack of improvements. They showed that if there is a subcubic combinatorial algorithm for triangle detection, then there is also a subcubic combinatorial algorithm for boolean matrix multiplication (BMM). Since any algorithm for computing minimum cycle in unweighted graphs can be used to detect triangles, any subcubic combinatorial algorithm for minimum cycle computation implies a subcubic combinatorial algorithm for BMM as well.

Vassilevska Williams and Williams [WW10] showed also that a subcubic algorithm for computing a minimum weight cycle in a weighted undirected graph with integral weights from the range $[1, M]$ implies a subcubic algorithm for computing all pair of shortest paths in a weighted directed graph with integral weights from the range $[-M, M]$. This implies that in order to obtain subcubic algorithms for computing minimum weight cycle the problem should be relaxed somehow. A possible relaxation that has been used extensively in the context of subcubic algorithms for computing shortest paths [ACIM99, DHZ00, CZ01, BGS05, BK06] is to compute an approximation of the exact solution. Indeed, recently Lingas and Lundell [LL09] presented two algorithms that approximate the minimum weight cycle. In particular, Lingas and Lundell presented a $2\frac{2}{3}$-approximation algorithm with $\tilde{O}(n^{3/2})$ running time (the $\tilde{O}$ order omits polylog factors from running time, so $\tilde{O}(n^{3/2}) = O(n^{3/2} polylog(n))$) for unweighted undirected graphs, and a 2-approximation algorithm with $O(n^2 \log n(\log n + \log M))$ running time for undirected graphs with integral weights from the range $[1, M]$. It is interesting to compare these results to almost quadratic algorithms for shortest paths approximation. Cohen and Zwick [CZ01] presented a 3-approximation algorithm with $\tilde{O}(n^2)$ running time for shortest paths in weighted undirected graphs. Baswana, Goyal and Sen [BGS05] and Baswana and Kavitha [BK06] obtained almost quadratic algorithms with stretch between 2 and 3. Dor, Halperin and Zwick [DHZ00] showed that using an algorithm that computes paths, with approximation less than 2 between all pairs of vertices in an undirected graph, it is possible to multiply two boolean matrices. This leads to the following open problem, that was posed by Lingas and Lundell in [LL09]: "Is it possible to obtain a subcubic algorithm with approximation less than 2 for minimum weight cycle". In this paper we provide an affirmative answer to this question. Given a weighted undirected graph $G = (V, E, w)$, $w : E \to \mathbb{R}^+$, let C be a minimum weight cycle of G, let $w(C)$ be the weight of C and let $w_{max}(C)$ be the weight of the maximal edge of C. For integral weights from the range $[1, M]$, we present an algorithm that reports a cycle of weight at most $4w(C)/3$ in $O(n^2 \log n(\log n + \log M))$ running time, which matches the running time of Lingas and Lundell. One part of this algorithm is an algorithm with the same running time, that reports a cycle of weight at most $w(C) + w_{max}(C)$. This algorithm can be viewed as a generalization for weighted graphs, of the algorithm of Itai and Rodeh [IR78] for unweighted graphs that has an additive error of one. We also present a strongly polynomial algorithm, that for any $\varepsilon > 0$, computes in $O(\frac{1}{\varepsilon}n^2 \log n(\log\log n))$ time a $(4/3 + \varepsilon)$-approximation of minimum weight cycle.

Our algorithms imply a real separation between minimum cycle approximation and shortest paths approximation. Moreover, the approximation value of 4/3 that we obtain is not accidental! Using the new result of Vassilevska Williams and Williams [WW10], we show that a $(4/3 - \varepsilon)$-approximation, where $0 < \varepsilon \leq 1/3$, implies a subcubic combinatorial algorithm for BMM. This implies that our 4/3-approximation algorithm is essentially optimal.

Our algorithms are surprisingly simple. The main idea is to classify cycles that are candidates to be a minimum weight cycle by their maximal edge (the heaviest edge on the cycle). A cycle whose maximal edge weight is relatively small has different properties than a cycle whose maximal edge weight is relatively large.

We find a threshold which above it we run an algorithm with a good approximation for a large maximal edge, and below it we run an algorithm with a good approximation for a small maximal edge. We will use a similar variant of Dijkstra's algorithm to the one used by Lingas and Lundell in [LL09]. Roughly speaking, Lingas and Lundell grow, using a bounded version of Dijkstra's algorithm, a tree from each vertex of the graph, and if the tree source is a vertex of the minimum cycle, then it produces a 2-approximation of the minimum cycle. The tree growth is controlled using the weight of the minimum cycle. In our algorithm, we use a different approach to control the tree growth, that is, we control the growth using parameters that depend also on the maximal edge weight. Moreover, we use the simple observation that there are several different vertices on the minimum weight cycle, and it is enough to detect the cycle only from one of these vertices.

The problem of computing a cycle with minimum weight is closely related to the decision problem, in which given a graph $G = (V, E)$ and an integer k, the algorithm decides whether G contains a simple cycle of length exactly k. Many variants of this problem were considered along the last two decades. (See [AYZ95, AYZ97, YZ94, YZ04]). Perhaps the most notable one is the seminal work of Alon, Yuster and Zwick [AYZ95], that presented an algorithm with $O(n^{\omega} \log n)$ running time for the problem, where ω is the exponent of fast matrix multiplication (the current best value of ω is $\omega < 2.376$ by Coppersmith and Don [CW90].)

The rest of this paper is organized as follows. In the next section we provide some preliminaries and a short description of the 2-approximation algorithm of Lingas and Lundell. In Section 3 we present the algorithm that reports a cycle of weight at most $w(C) + w_{max}(C)$, the algorithm that reports a cycle of weight at most $4w(C)/3$ and the hardness result. In Section 4 we present the $(4/3+\varepsilon)$-approximation with $\tilde{O}(n^2)$ running time.

2 Preliminaries

Let $G = (V, E, w)$ be a weighted undirected graph with non-negative edge weights. Let $w(u, v)$ be the weight of the edge (u, v). We define a cycle to be a set of vertices, ordered by their appearance on the cycle, starting at an arbitrary vertex. If $C = \{v_1, v_2, \ldots, v_\ell\}$ is a cycle in G, then $(v_i, v_{i+1}) \in E$, for every $i < \ell$ and $(v_\ell, v_1) \in E$. Let $w(C)$ be the sum of the weights of the edges of C, and let $w_{\max}(C)$ be the weight of the heaviest edge. We denote with $d_C[v_i, v_j]$ the weight of the path that traverses the cycle from v_i to v_j, by passing from v_i to v_{i+1} and so on. In the case that $j < i$, we traverse from v_ℓ to v_1, and continue until we reach v_j. Let $d_C[v_i, v_i] = w(C)$. During the paper we assume that for every vertex $u \in V$ its edges are ordered by their weight in a priority queue Q_u.

Lingas and Lundell [LL09] presented a 2-approximation algorithm for undirected graphs with integral weights from the range $[1, M]$ with a running time of $O(n^2 \log n(\log n + \log M)))$, that is, their algorithm computes a cycle with a weight of at most twice the weight of the minimum cycle in G. For the sake of completeness we provide here a short overview of their algorithm.

The weight of the minimum cycle is an integral value in the range $[1, nM]$. Lingas and Lundell [LL09] do a binary search over this interval for the smallest value for which their algorithm reports a cycle. For a given value, $t \in [1, nM]$, their algorithm performs, for each vertex $s \in V$ of the graph, a restricted shortest paths tree computation up to depth t, using an algorithm called BoundedDijkstra. Let $u, v \in V$, we denote $d[u, v]$ to be the current estimation of the distance of the shortest path from u to v over the algorithm execution. When a restricted shortest path starts from a given source vertex, $s \in V$, the short notation $d[v]$ will be used appropriately, instead of $d[s, v]$. In the regular implementation of Dijkstra's algorithm, each vertex is extracted from the priority queue at some stage and all its edges are relaxed. In the BoundedDijkstra algorithm of Lingas and Lundell [LL09], vertices are added to the priority queue, only if the current estimation on their distance from the source is at most t. More specifically, when a

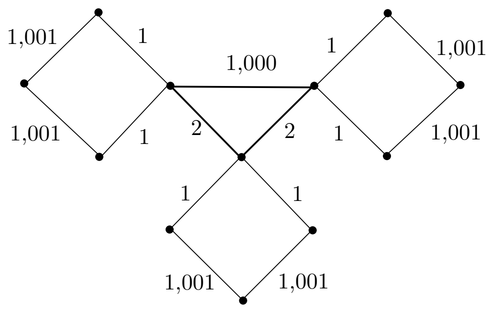

Figure 1: The triangle in the middle weights $1,004$. Lingas and Lundell algorithm returns a cycle that has a weight of $2,004$, which is almost twice than the weight of the minimum weight cycle in this graph.

Algorithm 1: BoundedDijkstra(G, s, t)

```
foreach $v \in V$ do $d[v] \leftarrow \infty$;
$d[s] = 0$;
$Q \leftarrow \{s\}$;
while $Q \neq \emptyset$ do
    $u \leftarrow$ Extract-Min$(Q)$;
    Controlled-Relax$(u, t)$;
```

Algorithm 2: Controlled-Relax(u, w_u)

```
$(u, v) \leftarrow$ Extract-Min$(Q_u)$;
while $d[u] + w(u, v) \leq w_u$ do
    RelaxOrStop$(u, v)$;
    $(u, v) \leftarrow$ Extract-Min$(Q_u)$;
```

Algorithm 3: RelaxOrStop(u, v)

```
if $d[v] \neq \infty$ then
    report a cycle and stop else Relax$(u, v)$;
```

vertex u is extracted from the priority queue, its edges are relaxed in a non-decreasing order of weight, by extracting the minimal edge from Q_u at each step, until an edge (u, v) with $d[u] + w(u, v) > t$ is reached. Before an edge (u, v) is relaxed, they check whether $d[v]$ is already set, meaning, the vertex v has already got an estimation by the algorithm to its distance from source vertex s. If $d[v]$ is already set, then there is a path from s to v that uses some other edge rather than (u, v) to reach v, and its weight is at most t. Hence, there is a cycle, and its total weight is at most $2t$. The algorithm BoundedDijkstra is presented in Algorithm 1. (Notice that the presentation is biased towards our later improvements.) Given a vertex u, that is extracted from the priority queue Q, its edges are relaxed by using Controlled-Relax(u, t). Cycle detection and relaxations are done in RelaxOrStop(u, v). If a cycle is detected before the edge (u, v) is relaxed, then it can be reported by traversing the shortest paths tree to the least common ancestor of u and v. The cost of that is proportional to the number of edges on the cycle, which is less than n. It is quite straightforward to see that for every $v \in V$ the value of $d[v]$ is set only once, thus the cost of running BoundedDijkstra for a given vertex is $O(n \log n)$, and the total cost for all the vertices is $O(n^2 \log n)$.

As we mentioned above, the right value of t is not known in advance, and a binary search is done over the possible values of the minimum cycle in the range $[1, nM]$ for the smallest value of t, for which the algorithm reports a cycle. There are $O(\log n + \log M)$ steps in the binary search. We conclude that the total cost of the algorithm of Lingas and Lundell [LL09] is $O(n^2 \log n(\log n + \log M))$. We also mention that the 2-approximation algorithm of Lingas and Lundell is indeed tight, the illustration in Figure 1 give an example of a graph that an execution of Lingas and Lundell algorithm returns a cycle that is almost twice larger than the minimum weight cycle in the graph.

3 A 4/3-approximation of the minimum cycle

In this section we present a 4/3-approximation algorithm to the minimum weight cycle in a weighted undirected graph with integral weights from the set $[1, M]$. The running time is $O(n^2 \log n(\log n + \log M))$. We use the same framework as Lingas and Lundell [LL09], that is, we search for the weight of the minimum cycle using binary search over the interval $[1, nM]$. However, for each of the possible values, we run an algorithm which guarantees an approximation of 4/3 to the weight of the minimum cycle. Our algorithm is a hybrid algorithm that combines two different algorithms to obtain the desired approximation. Let C be a minimum weight cycle. The first algorithm that we present reports a cycle of weight at most $w(C) + w_{\max}(C)$. An interesting property of this algorithm is that it produces the cycle without knowing the value of $w_{\max}(C)$. The second algorithm that we present gets two additional parameters w_l and w_u, and if $w_{\max}(C) \in [w_l, w_u]$ it reports a cycle of weight at most $\max\{2w(C) - 2w_l, w(C) + w_u - w_l\}$. We then present the hybrid algorithm, that combines the two algorithms in an optimal way to obtain 4/3 approximation. We end this section with our hardness result.

3.1 Approximating the minimum cycle when the maximal edge is small

Here we present an $O(n^2 \log n)$ time algorithm that reports a cycle of weight at most $w(C) + w_{\max}(C)$. The algorithm does not need to know the value of $w_{\max}(C)$. The algorithm works independently from each vertex of the graph. We show that there exists a vertex $s \in C$, such that when the algorithm runs from s, it reports a cycle of weight at most $w(C) + w_{\max}(C)$. Through this section we use t to denote the minimum cycle weight, that is, $w(C) = t$.

The main idea of the algorithm is to use the following simple property of cycles. Given a cycle, there exists an edge, for every vertex s of the cycle, such that both its endpoints are at a distance of at most $t/2$ from s. (See an illustration in Figure 3.1). We first prove this property, and then describe how our algorithm exploits it.

Lemma 1. *Let $C = \{v_1, \ldots, v_\ell\}$ be a cycle and let $w(C) = t$. For every vertex $s \in C$ there exist two vertices $v_i, v_{i+1} \in C$, such that $d[s, v_i] \leq t/2$ and $d[s, v_{i+1}] \leq t/2$*

Proof. Let (v_i, v_{i+1}) be an edge of C and let $w(v_i, v_{i+1}) \geq t/2$. It is easy to see that in this case, for every $s \in C$, the edge (v_i, v_{i+1}) satisfies the claim. Thus, we can assume $w_{\max}(C) < t/2$. Since the proof is for every $s \in C$ and the indexing is arbitrary we can assume, wlog, that $s = v_1$. We then start to traverse the cycle by increasing the index by one at each step until we reach a vertex v_i that satisfies $d_C[v_1, v_i] \leq t/2$ and $d_C[v_1, v_{i+1}] > t/2$. The vertices v_i and v_{i+1} satisfy that $d[v_1, v_i] \leq d_C[v_1, v_i] \leq t/2$ and $d[v_1, v_{i+1}] \leq d_C[v_{i+1}, v_1] \leq t/2$ as required. We only need to show that such vertices are found by the process we described. The weight of every cycle edge is strictly less than $t/2$, thus $d_C[v_1, v_2] < t/2$. If $d_C[v_1, v_3] \leq t/2$, we continue to v_3. Each time we go from v_k to v_{k+1} it is because $d_C[v_1, v_{k+1}] \leq t/2$. There are two possible cases. One is that eventually we reach to vertices v_k and v_{k+1} such that $d_C[v_1, v_k] \leq t/2$, $d_C[v_1, v_{k+1}] > t/2$ and $k < \ell$ as required. The other case is that we reach v_ℓ without finding a vertex that satisfies the condition. But this cannot happen as $d_C[v_1, v_\ell] \leq t/2$ implies that $w(v_1, v_\ell) \geq t/2$, which contradicts the assumption that $w_{\max}(C) < t/2$. □

We now turn to describe the algorithm. It has two steps. In the first step it discovers all the vertices at a distance of at most $t/2$ from s. Let u be a vertex that is extracted at this stage. We relax only edges $(u, v) \in E$, if $d[s, u] + w(u, v) \leq t/2$. For each such an edge (u, v), we check before we relax it if $d[v] \neq \infty$, and if it is we report a cycle and stop the algorithm. This is done by calling Controlled-Relax$(u, t/2)$. Let (u, v') be the smallest edge of u that was not relaxed, that is, $d[s, u] + w(u, v') > t/2$. We add (u, v') to a special priority queue R with the key $d[s, u] + w(u, v')$. (We can obtain (u, v') by keeping the last edge

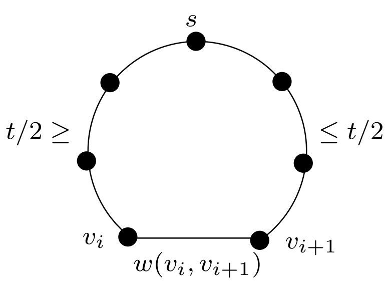

Figure 2: The vertex s and the edge (v_i, v_{i+1}) that satisfies the property.

extracted from Q_u.) The first step ends when Q gets empty. The first step is depicted in Figure 3. We then pass to the second step, in which we carefully pick edges to relax. More specifically, in this step we use the priority queue R to relax edges. In each iteration we extract the minimal edge of R and relax it. Let (u, v) be an edge extracted from R. We check if $d[v] \neq \infty$, and if it is we report on a cycle and stop the algorithm. If $d[v] = \infty$, then we relax the edge (u, v) and add to R the smallest edge that touches u that was not relaxed yet. The second step ends either when a cycle is detected, or when R becomes empty. The algorithm is presented in Algorithm 5. We now prove its correctness.

Lemma 2. *Let $C = \{v_1, \ldots, v_\ell\}$ be a cycle and let $w(C) = t$. The algorithm Approx-Shortest-Cycle(G, s, t) returns a cycle of weight at most $t + w_{\max}(C)$ for every $s \in C$.*

Proof. Consider the iteration of the algorithm for $s \in C$. From Lemma 1, it follows that there exist two vertices v_i and v_{i+1}, such that $d[s, v_i] \leq t/2$ and $d[s, v_{i+1}] \leq t/2$. The first stage of the algorithm ends either because a cycle is found or because Q gets empty. If a cycle is reported in the first stage, then its weight is at most t. This follows from the fact that during this stage an edge (u, v) is considered for relaxation only if $d[s, u] + w(u, v) \leq t/2$, and if (u, v) is not relaxed since $d[v]$ is already set, then $d[v] \leq t/2$ and there must be a cycle of weight at most t. We can output a simple cycle by traversing the shortest paths tree backward from v and from u. We now assume that no cycle is reported in the first stage. This means that both v_i and v_{i+1} were extracted from the priority queue Q during the first stage, because $d[s, v_i] \leq t/2$ and $d[s, v_{i+1}] \leq t/2$ and all vertices of distance at most $t/2$ are extracted during the first stage. Moreover, because we know that there exists an edge $(v_i, v_{i+1}) \in E$ there must be two edges (v_i, x) and (v_{i+1}, y) that were added in the first stage to R. (Notice that it might be that $(v_i, x) = (v_{i+1}, y) = (v_i, v_{i+1})$, but we cannot guarantee that.) The key of (v_i, x) is at most $d[s, v_i] + w(v_i, v_{i+1})$, and the key of (v_{i+1}, y) is at most $d[s, v_{i+1}] + w(v_i, v_{i+1})$. The second step of the algorithm can end either when a cycle is detected, or when R becomes empty. We show that a cycle of weight at most $t + w_{\max}(C)$ is detected. Assume, wlog, that $d[s, v_i] \leq d[s, v_{i+1}]$. If no cycle is found before the edge (v_i, v_{i+1}) is extracted from R, then a cycle of weight at most t is detected. Thus, we can assume that a cycle is detected before the edge (v_i, v_{i+1}) is extracted from R. Let (u, v) be the edge that is relaxed immediately following a cycle detection. Since (v_i, v_{i+1}) is not relaxed yet (and even might not be in the priority queue R yet), the key of (u, v) is at most $d[s, v_i] + w(v_i, v_{i+1})$, which implies that $d[s, u] + w(u, v) \leq d[s, v_i] + w(v_i, v_{i+1})$. Since a cycle is detected, then $d[v] \neq \infty$. If $d[v]$ was set during the first stage, then $d[v] \leq t/2$, and a cycle of weight at most $d[s, v_i] + w(v_i, v_{i+1}) + t/2$ is detected. From Lemma 1 it follows that $d[s, v_i] \leq t/2$, hence:

$$d[s, v_i] + w(v_i, v_{i+1}) + t/2 \leq t/2 + w(v_i, v_{i+1}) + t/2 = t + w(v_i, v_{i+1}) \leq t + w_{\max}(C).$$

If $d[v]$ was set only in the second stage, it was set before the relaxation of (u, v), which implies that $d[v] \leq d[s, v_i] + w(v_i, v_{i+1})$, and a cycle of weight at most $2(d[s, v_i] + w(v_i, v_{i+1}))$ is detected. Since the indexing is arbitrary we can assume $s = v_1$. Since $d[s, v_i] \leq d_C[s, v_i]$, $d[s, v_{i+1}] \leq d_C[v_{i+1}, s]$ and $d[s, v_i] \leq d[s, v_{i+1}]$ we get:

$$2(d[s, v_i] + w(v_i, v_{i+1})) \leq d_C[s, v_i] + 2w(v_i, v_{i+1}) + d_C[v_{i+1}, s] = t + w(v_i, v_{i+1}) \leq t + w_{\max}(C).$$

Algorithm 4: Approx-Shortest-Cycle-S(G, t)

```
$C^* \leftarrow \emptyset$;
foreach $s \in V$ do
    $C' \leftarrow$ Approx-Shortest-Cycle($G, s, t$);
    if $w(C') < w(C^*)$ then $C^* \leftarrow C'$
return $C^*$
```

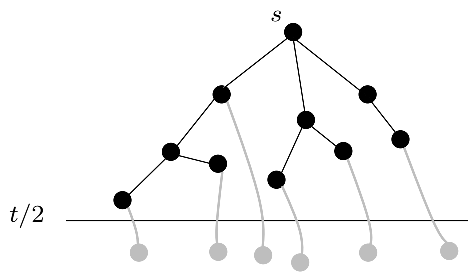

Figure 3: The tree after the first stage. The thick grey edges are added to R.

Algorithm 5: Approx-Shortest-Cycle(G, s, t)

```
foreach $v \in V$ do $d[v] \leftarrow \infty$;
$d[s] = 0$;
$R \leftarrow \emptyset$;
$Q \leftarrow \{s\}$;
while $Q \neq \emptyset$ do
    $u \leftarrow$ Extract-Min($Q$);
    Controlled-Relax($u, t/2$);
    Let $(u, v)$ be the smallest edge of $u$ that was not relaxed;
    add $(u, v)$ to $R$ with key $d[u] + w(u, v)$;
while $R \neq \emptyset$ do
    $(u, v) \leftarrow$ Extract-Min($R$);
    RelaxOrStop($u, v$);
    $(u, v) \leftarrow$ Extract-Min($Q_u$);
    add $(u, v)$ to $R$ with key $d[u] + w(u, v)$;
```

□

We now turn to prove that the running time of the algorithm Approx-Shortest-Cycle(G, s, t) is $O(n \log n)$.

Lemma 3. *Let $G = (V, E, w)$ be a weighted undirected graph and let $s \in V$ be an arbitrary vertex. The algorithm Approx-Shortest-Cycle(G, s, t) stops after $O(n \log n)$ time.*

Proof. The algorithm can stop in three different situations. It can stop in the first stage, when a cycle is detected. It can stop in the second stage, when a cycle is detected, and if it does not stop in neither of these cases, then it stops in the second stage, when the priority queue R becomes empty. We first analyze the cost of the first stage. Let u be a vertex that is being extracted from Q during the first stage. Let (u, v) be an edge that is about to be relaxed. If $d[v] \neq \infty$, the algorithm stops immediately and reports a cycle. This implies that we only relax (u, v) if $d[v] = \infty$, hence, we can charge the relaxation of (u, v) to v. Notice that the algorithm will not relax any other edge (u', v), since before the edge (u', v) is about to be relaxed the algorithm stops and reports a cycle. Let (u, v') be the smallest edge of u that is not relaxed. The algorithm adds (u, v') to R. We charge the cost of this addition to u. We conclude that if a cycle is not reported during the first stage, then the cost of this stage is at most $O(n \log n)$, since any vertex is charged at most twice. If a cycle is reported, then it only happens once, so we can charge it to the vertex s, and the cost is still $O(n \log n)$.

We now turn to the second stage. We can ignore edges that are extracted from R and were added to it at the first stage, since their cost was already charged in the first stage. Thus, we focus on edges that are added to R during the second stage. When we add a new edge (u, v') to R, then it must be that the last edge that was extracted from R is an edge (u, v), such that $d[v] = \infty$ before (u, v) is relaxed, as otherwise a cycle is reported. We will charge v with the cost of the addition of (u, v') to R. Notice that we cannot charge (u, v') to v', since many edges that are added to R might end at v'. However, when an edge that ends at v will be extracted from R again, then before the algorithm relaxes this edge, it stops and reports a cycle since $d[v] \neq \infty$. Thus, using this charging, v will be charged only for the edge (u, v') during the second stage. We conclude that the cost of this stage is $O(n \log n)$ as well. □

Algorithm 6: Approx-Shortest-Cycle-L(G, t, w_l, w_u)

```
$C^* \leftarrow \emptyset$;
**foreach** $s \in V$ **do**
    **foreach** $v \in V$ **do** $d[v] \leftarrow \infty$;
    $d[s] = 0$;
    $Q \leftarrow \emptyset$;
    Controlled-Relax$(s, w_u)$;
    **while** $Q \neq \emptyset$ **do**
        $u \leftarrow$ Extract-Min$(Q)$;
        Controlled-Relax$(u, t - w_l)$;
        **if** *a cycle $C'$ is reported* **then**
            **if** $w(C') < w(C^*)$ **then** $C^* \leftarrow C'$
**return** $C^*$;
```

We end this section by showing how to detect a cycle of length at most $t + w_{\max}(C)$ in $O(n^2 \log n)$ time. The algorithm is given in Algorithm 4. We run Approx-Shortest-Cycle(G, s, t) from every vertex of the graph, and return the minimal cycle that is detected.

Theorem 4. *Let $G = (V, E, w)$ be a weighted undirected graph. Let C be a minimum weight cycle and let $w(C) = t$. Algorithm Approx-Shortest-Cycle-S(G, t) reports a cycle of weight at most $t + w_{\max}(C)$ in time $O(n^2 \log n)$ without knowing $w_{\max}(C)$.*

Proof. Algorithm Approx-Shortest-Cycle-S(G, t) runs algorithm Approx-Shortest-Cycle(G, s, t) for every vertex $s \in V$. From Lemma 3, we know that the cost of running Approx-Shortest-Cycle(G, s, t) from a specific vertex s is $O(n \log n)$, thus the total cost of Approx-Shortest-Cycle-S(G, t) is $O(n^2 \log n)$. Moreover, every vertex $s \in C$ will be the source of one run of Approx-Shortest-Cycle(G, s, t). From Lemma 2, we know that in such a running a cycle of weight at most $t + w_{\max}(C)$ is reported without knowing $w_{\max}(C)$. Since Approx-Shortest-Cycle-S(G, t) reports the minimum among all the cycles reported, it is guaranteed that a cycle of weight at most $t + w_{\max}(C)$ will be reported. □

3.2 Approximating the minimum cycle when the maximal edge is large

Let $G = (V, E, w)$ be a weighted undirected graph and let C be a minimum weight cycle. Let $w(C) = t$ and let $w_{\max}(C) \in [w_l, w_u]$, that is, w_l (w_u) is a lower (upper) bound on the maximal edge weight. We present an algorithm that reports a cycle of weight at most $\max\{2t - 2w_l, t + w_u - w_l\}$ in $O(n^2 \log n)$ time.

The algorithm iterates on the vertices of the graph. For each vertex $s \in V$ it has two stages. In the first stage, all edges of s whose weights are at most w_u are relaxed. In the second stage, we grow from s a shortest paths tree up to depth $t - w_l$, using the priority queue Q that was initialized with the other endpoints of the edges of s, that were relaxed in the first stage. Notice that in the second stage we relax an edge (u, v) only if $d[u] + w(u, v) \leq t - w_l$. If s is an endpoint of a maximal edge of C, then in the second stage of the algorithm, a cycle of weight at most $\max\{2t - 2w_l, t + w_u - w_l\}$ is reported. The algorithm is presented in Algorithm 6. Next, we prove the correctness of the algorithm.

Lemma 5. *Let $C = \{v_1, \ldots, v_\ell\}$ be a cycle of weight t and let $w_l \leq w_{\max}(C) \leq w_u$. The algorithm Approx-Shortest-Cycle-L(G, t, w_l, w_u) returns a cycle of weight at most $\max\{2t - 2w_l, t + w_u - w_l\}$.*

Proof. Let (v_{i-1}, v_i) be an edge of maximum weight in C and let $w(v_{i-1}, v_i) \in [w_l, w_u]$. Consider the iteration of the algorithm in which v_i is the source. Recall that in this iteration we relax all edges of v_i of weight at most w_u, and grow a shortest paths tree up to depth $t-w_l$. Since (v_{i-1},v_i) is a maximal edge, it follows that $w(v_{i-1},v_i) \geq w(v_i,v_{i+1})$. We now distinguish between two cases. For the first case, assume that there is a vertex $v_k \in C$, such that the shortest path between v_i and v_k is not using the edge (v_i,v_{i+1}), and let v_k be the first vertex on the cycle starting from v_{i+1}, in increasing order, that satisfies this requirement. Let $P = \{v_i, v_{i+1}, v_{i+2}, \ldots, v_{k-1}, v_k\}$ be the portion of C from v_i to v_k. We know that $d_C[v_i,v_k] \leq t-w(v_{i-1},v_i) \leq t-w_l$. Thus, when the edge (v_{k-1},v_k) is about to be relaxed $d[v_{k-1}]+w(v_{k-1},v_k) \leq t-w_l$. Let $P' = \{v_i, x_1, x_2, \ldots, x_j, v_k\}$ be the shortest path between v_i and v_k. Its weight is bounded by $t-w_l$. Thus, when the edge (x_j,v_k) is about to be relaxed $d[x_j]+w(x_j,v_k) \leq t-w_l$. This implies that there are two different paths from v_i to v_k, both of weight at most $t-w_l$, and a cycle of weight at most $2t-2w_l$ is reported, unless some other cycle was reported earlier. We now turn to the second case, in which the shortest path between v_i and each of the vertices of C uses the edge (v_i,v_{i+1}). In particular, the shortest path from v_i to v_{i-1} uses (v_i,v_{i+1}) and not (v_{i-1},v_i). Let $P = \{v_i, v_{i+1}, v_{i+2}, \ldots, v_{i-2}, v_{i-1}\}$. When the edge (v_{i-2},v_{i-1}) is about to be relaxed $d[v_{i-2}]+w(v_{i-2},v_{i-1}) \leq t-w_l$. Moreover, in the first stage of the algorithm, we relaxed the edge (v_{i-1},v_i), thus $d[v_{i-1}] \leq w(v_{i-1},v_i)$. We conclude that if a cycle is not reported before, then one of the cycles described above will be reported.

The cycle C satisfies one of the two cases described above. However, it might be that some other cycle will be reported earlier. We now bound the weight of such a cycle. A cycle can be reported only when an edge (u,v) is about to be relaxed and $d[v] \neq \infty$. The value of $d[v]$ is bounded either by w_u, if $d[v]$ was set in the first stage, or by $t-w_l$ if $d[v]$ was set in the second stage. In both cases, the algorithm intended to relax (u,v), which means that there is another path from s to v with length of at most $t-w_l$. This implies that there is a cycle of weight at most $\max\{2t-2w_l, t+w_u-w_l\}$.

□

We are now ready to prove the main Theorem of this section.

Theorem 6. *Let $G=(V,E,w)$ be a weighted undirected graph with minimum cycle C. Let $w(C)=t$ and $w_{max}(C) \in [w_l,w_u]$. Algorithm Approx-Shortest-Cycle-L(G,t,w_l,w_u) reports a cycle of weight at most $\max\{2t-2w_l, t+w_u-w_l\}$ in time $O(n^2 \log n)$.*

Proof. From Lemma 5 it follows that a cycle with the desired approximation is reported by the algorithm when the endpoints of the maximal edge of C are considered. The algorithm iterates on the vertices of the graph. The cost of a single iteration for a given vertex s is composed from relaxing the edges of s, whose weight is at most w_u in the first stage, and growing a bounded shortest paths tree to detect a cycle in the second stage. The cost of the first stage is at most $O(n \log n)$, since s has at most n edges. The cost the second stage is $O(n \log n)$, as we have shown in the previous section. We conclude that the running time of the algorithm is $O(n^2 \log n)$. □

3.3 The hybrid algorithm with 4/3-approximation

In this section we show how to obtain a 4/3-approximation algorithm using the two algorithms described above. The first algorithm reports a cycle of weight $w(C)+w_{\max}(C)$. The second algorithm reports a cycle of weight $\max\{2t-2w_l, t+w_u-w_l\}$, where $w(C)=t$ and $w_l \leq t \leq w_u$. Since the first algorithm guarantees its bound without knowing the actual value of $w_{\max}(C)$ we can always run it. However, as the maximal edge weight increases its approximation becomes worse. This implies that in order to get the best possible approximation we must use in some stage the second algorithm. The main question is how to set the parameters of the second algorithm, that is, from which maximal edge weight value and on it is better to use the second algorithm rather than the first algorithm. Let w^* be this weight. Since w^* is a transition point between the algorithms, both must have the same approximation bound for w^*. Thus, we

Algorithm 7: Approx-Shortest-Cycle$_{4/3}(G, t)$

```
$C_1 \leftarrow C_2 \leftarrow C_3 \leftarrow \emptyset$;
$C_1 \leftarrow$ Approx-Shortest-Cycle-S$(G, t)$;
**if** $w(C_1) > 4t/3$ **then** $C_1 \leftarrow \emptyset$;
$C_2 \leftarrow$ Approx-Shortest-Cycle-L$(G, t, t/3, 2t/3)$;
$C_3 \leftarrow$ Approx-Shortest-Cycle-L$(G, t, 2t/3, t)$;
**return** $\min(C_1, C_2, C_3)$;
```

can compute the value of w^*, by solving the equation $t + w^* = 2t - 2w^*$. We get that $w^* = t/3$ and set w_l to be $t/3$. We run the first algorithm, and uses the cycle that it reports, only if its weight is at most $4t/3$. The approximation of the second algorithm is $\max\{2t - 2w_l, t + w_u - w_l\}$. We set $w_l = t/3$ and balance the two expressions of the bound to obtain the value of w_u. We get that $2t - 2 \cdot t/2 = t + w_u - t/3$ and $w_u = 2t/3$. So if $w_{\max}(C) \leq 2t/3$, we have 4/3 approximation. We now turn to the case that $2t/3 < w_{\max}(C) \leq t$. In this case the bound of the second algorithm is dominated by $t + w_u - w_l$. If we set $w_l = 2t/3$ and $w_u = t$, we cover the desired range with 4/3 approximation as well. The algorithm is presented in Algorithm 7. We conclude this section with the following Theorem:

Theorem 7. *Let $G = (V, E, w)$ be a weighted undirected graph with integral weights from $[1, M]$. It is possible to compute in $O(n^2 \log n(\log n + \log M))$ time a 4/3-approximation of the minimum cycle of G.*

Proof. Let C be a minimum weight cycle and let $w(C) = t^*$. If $w_{\max}(C) \leq t^*/3$, it follows from Theorem 4, that Approx-Shortest-Cycle-S(G, t^*) reports a cycle of weight $4t^*/3$. If $t^*/3 \leq w_{\max}(C) \leq 2t^*/3$, it follows from Theorem 6, that Approx-Shortest-Cycle-L$(G, t^*, t^*/3, 2t^*/3)$ reports a cycle of weight $4t^*/3$. Finally, if $2t^*/3 \leq w_{\max}(C) \leq t^*$, it follows from Theorem 6, that Approx-Shortest-Cycle-L$(G, t^*, 2t^*/3, t^*)$ reports a cycle of weight $4t^*/3$ as well. This implies that Approx-Shortest-Cycle$_{4/3}(G, t^*)$ always reports a cycle of weight at most $4t^*/3$. Moreover, for any $t' > t^*$ Approx-Shortest-Cycle$_{4/3}(G, t')$ always reports a cycle of weight at most $4t'/3$.

This implies that we can do a binary search over the interval $[1, nM]$ for a value t, such that Approx-Shortest-Cycle$_{4/3}(G, t)$ reports a cycle and Approx-Shortest-Cycle$_{4/3}(G, t-1)$ does not. The search is done as follows. Let $[I, J]$ be the interval we currently consider. We search for a cycle of weight at most $(J + I)/2$. If a cycle is reported, we consider in the next step the interval $[I, \lfloor (J + I)/2 \rfloor]$, and if not we consider in the next step the interval $[\lceil (J + I)/2 \rceil, J]$. Let t be the value found by the search, and let $\hat{t}$ be the weight of the cycle reported by Approx-Shortest-Cycle$_{4/3}(G, t)$. Since, according to the binary search, t is the minimal value of which a cycle is detected, and since we know that for t^* Approx-Shortest-Cycle$_{4/3}(G, t^*)$ must report a cycle of weight at most $4t^*/3$, it follows that $t \leq t^*$, which implies that $\hat{t} \leq 4t/3 \leq 4t^*/3$. The running time of the algorithm is $O(n^2 \log n(\log n + \log M))$, since each step of the search takes $O(n^2 \log n)$ time and there are at most $O(\log n + \log M)$ steps. □

3.4 Subcubic equivalences between BMM and minimum cycle approximation

We end this section by showing that the 4/3 approximation that we have obtained is essentially optimal. In a recent breakthrough Vassilevska Williams and Williams [WW10] showed that many important problems on graphs and matrices, that have an $O(n^3)$ time algorithm, are equivalent under subcubic reductions. They defined an algorithm for n nodes graph with integral edge weights from $[-M, M]$ to be truly subcubic, if it runs in $O(n^{3-\delta}\text{poly}(\log M))$ for some $\delta > 0$. Among their results they showed a subcubic equivalence between boolean matrix multiplication (BMM) and triangle detection in undirected graphs. We use this equivalence to show that any algorithm, that reports a cycle whose weight is a multiplicative approximation of $4/3 - \varepsilon$ for the minimum cycle weight, implies a truly subcubic algorithm for BMM.

Theorem 8. *If there is a truly subcubic combinatorial algorithm, that approximates the minimum cycle in an unweighted undirected graph with a multiplicative approximation and that is strictly less than $4/3$, then there is a truly subcubic combinatorial algorithm for BMM.*

Proof. Vassilevska Williams and Williams [WW10] showed that BMM is equivalent to triangle detection. We will show that if there exists a truly subcubic algorithm, that reports a cycle whose weight is an approximation of the minimum cycle weight with a multiplicative error of $4/3 - \varepsilon$, for some $\varepsilon > 0$, then it is possible to detect a triangle. Let $G = (V, E)$ be unweighted undirected graph that contains a triangle. The minimum cycle of G is of weight 3. All cycles of G that are not minimal are of weight at least 4. In such a case, an algorithm that reports a $4/3 - \varepsilon$ approximation of the minimum cycle, can report only on cycles of weight 3, since any other cycle is of weight at least 4, and hence violates the approximation bound. If G does not contain a triangle, then the algorithm reports a cycle of weight at least 4. This implies that using the $(4/3 - \varepsilon)$-approximation algorithm, for the minimum cycle problem, we can detect a triangle, and hence multiply two boolean matrices. □

4 A strongly polynomial algorithm with $(4/3 + \varepsilon)$-approximation

In this section we present an algorithm, that for any $\varepsilon > 0$, computes a $(4/3 + \varepsilon)$-approximation in $O(n^2 \log n \cdot \log_{1+\varepsilon} \log n)$ time. The graph in this case may have non-negative real edge weights. We first present an algorithm, that provides a $\log n$ multiplicative approximation for the weight of the minimum cycle in $O(n^2 \log n)$ time. We then show that using this rough approximation it is possible to avoid the binary search step of the previous algorithm, and to obtain $(4/3 + \varepsilon)$-approximation in $O(n^2 \log n \cdot \log_{1+\varepsilon} \log n)$ time.

Before we turn to describe the $\log n$ approximation, we need to define the notion of *spanners*.

Definition 9 (Spanners [PS89]). *Let $G = (V, E, w)$ be a weighted undirected graph, and let $k \geq 1$. A subgraph $H = (V, E')$ is said to be a k-spanner of G, if and only if, for every $u, v \in V$ we have $d_H(u, v) \leq k \cdot d_G(u, v)$ and $E' \subseteq E$.*

Althöfer et al. [ADD+93] presented a greedy algorithm for constructing sparse spanners of weighted undirected graphs. For every integer $k \geq 2$, it constructs a $(2k - 1)$-spanner with at most $n^{1+1/k}$ edges. This is an essentially optimal tradeoff between stretch and size. The algorithm is reminiscent of Kruskal's algorithm for the construction of a minimum spanning tree algorithm. A naive implementation of this algorithm requires $O(mn^{1+1/k})$ time. Roditty and Zwick [RZ04] showed that it is possible to implement a modified version of the greedy algorithm with the same space and stretch tradeoff, so that it will run in $O(kn^{2+1/k})$ time.

Let $G = (V, E, w)$ be a weighted undirected graph. The $\log n$ approximation algorithm first computes a spanner H of G with $k = \log n$. The cost of computing H, using Roditty and Zwick [RZ04] spanner construction, is $O(n^2 \log n)$. The resulted spanner H is a $(\log n)$-spanner with $O(n)$ edges. For every edge $(u, v) \in E$, it follows from the spanner definition that $d_H(u, v) \leq \log n \cdot w(u, v)$. This implies that the weight of the minimum cycle of H is a $\log n$ approximation for the weight of the minimum cycle of G. The minimum cycle of H can be computed using the trivial algorithm for computing a minimum cycle, and since H contains only $O(n)$ edges, the cost of that is $O(n^2 \log n)$. We now show how using this rough approximation of $\log n$ we can efficiently obtain an approximation of $4/3 + \varepsilon$.

Assume that the weights of the edges are real numbers from the interval $[0, M]$. We first detect zero weight cycles, by removing every non-zero edge from the graph and checking whether the remaining graph contains a cycle in $O(m)$ time. If a cycle exists, then it is a minimum cycle, so we report it and stop. If there is no zero weight cycle, it implies that the weight of the minimum cycle is a real value from the interval $(0, nM]$.

Let c be the smallest non-zero edge weight in G. We scale the edge weights such that the smallest edge weight is 1, by multiplying the weight of every edge by $1/c$. We get that the weight of the minimum cycle is in the real interval $[1, nM']$, where $M' = M/c$. We divide the interval $[1, nM']$ as follows:

$$[1, 1+\varepsilon), [1+\varepsilon, (1+\varepsilon)^2), [(1+\varepsilon)^2, (1+\varepsilon)^3), \cdots, [(1+\varepsilon)^{\log_{1+\varepsilon}(nM')-1}, (1+\varepsilon)^{\log_{1+\varepsilon}(nM')}]$$

We now use the $\log n$ approximation to obtain a $(4/3+\varepsilon)$-approximation for every $\varepsilon > 0$. Let t^* be the weight of the minimum cycle in G and let $\hat{t}$ be the weight of the minimum cycle in H. There exist i, j and k such that $i \le j \le k$ and

$$\hat{t}/\log n \in [(1+\varepsilon)^i, (1+\varepsilon)^{i+1}) \quad t^* \in [(1+\varepsilon)^j, (1+\varepsilon)^{j+1}) \quad \hat{t} \in [(1+\varepsilon)^k, (1+\varepsilon)^{k+1})$$

We run Approx-Shortest-Cycle$_{4/3}(G,t)$ for every $t \in \{(1+\varepsilon)^i, (1+\varepsilon)^{i+1}, (1+\varepsilon)^{i+2}, \cdots, (1+\varepsilon)^{k+1}\}$ and report the shortest cycle found. It is easy to see that this cycle is a $(\frac{4}{3} \cdot (1+\varepsilon))$-approximation to the minimum cycle. It is only left to show that the number of times that we run Approx-Shortest-Cycle$_{4/3}(G,t)$ is bounded by $O(\log_{1+\varepsilon} \log n)$. We run it $k-i+2$ times. We know that $\hat{t} \le \log n(1+\varepsilon)^{i+1}$ and that $(1+\varepsilon)^k \le \hat{t}$. Combining the two inequalities we get that $k-i-1 \le \log_{1+\varepsilon} \log n$, hence, the number of times we run Approx-Shortest-Cycle$_{4/3}(G,t)$ is $O(\log_{1+\varepsilon} \log n)$. The next Theorem stems from the above discussion.

Theorem 10. *Let $G = (V, E, w)$ be a weighted undirected graph with non-negative real edge weights. For any $\varepsilon > 0$, we can compute in $O(\frac{1}{\varepsilon} n^2 \log n \cdot \log\log n)$ time, a cycle whose weight is a $(4/3+\varepsilon)$-approximation of the weight of the minimum weight cycle in G.*

References

[ACIM99] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999.

[ADD+93] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993.

[AYZ95] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[AYZ97] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[BGS05] S. Baswana, V. Goyal, and S. Sen. All-pairs nearly 2-approximate shortest-paths in o(n^2 polylog n) time. In V. Diekert and B. Durand, editors, *STACS*, volume 3404 of *Lecture Notes in Computer Science*, pages 666–679. Springer, 2005.

[BK06] S. Baswana and T. Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *FOCS*, pages 591–602. IEEE Computer Society, 2006.

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

[CZ01] E. Cohen and U. Zwick. All-pairs small-stretch paths. *J. Algorithms*, 38(2):335–353, 2001.

[DHZ00] D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM J. Comput.*, 29(5):1740–1759, 2000.

[IR78] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978.

[LL09] A. Lingas and E. M. Lundell. Efficient approximation algorithms for shortest cycles in undirected graphs. *Inf. Process. Lett.*, 109(10):493–498, 2009.

[PS89] D. Peleg and A.A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13:99–116, 1989.

[RZ04] L. Roditty and U. Zwick. On dynamic shortest paths problems. In *Proc. of 12th ESA*, pages 580–591, 2004.

[WW10] V. V. Williams and R. Williams. Subcubic equivalences between path, matrix, and triangle problems. In *FOCS*, 2010.

[YZ94] R. Yuster and U. Zwick. Finding even cycles even faster. In S. Abiteboul and E. Shamir, editors, *ICALP*, volume 820 of *Lecture Notes in Computer Science*, pages 532–543. Springer, 1994.

[YZ04] R. Yuster and U. Zwick. Detecting short directed cycles using rectangular matrix multiplication and dynamic programming. In J. I. Munro, editor, *SODA*, pages 254–260. SIAM, 2004.