A shortest cycle for each vertex of a graph

Raphael Yuster
Department of Mathematics
University of Haifa
Haifa 31905, Israel
E–mail: raphy@math.haifa.ac.il

Abstract

We present an algorithm that finds, for each vertex of an undirected graph, a shortest cycle containing it. While for directed graphs this problem reduces to the All-Pairs Shortest Paths problem, this is not known to be the case for undirected graphs.

We present a truly sub-cubic randomized algorithm for the undirected case. Given an undirected graph with n vertices and integer weights in $1, \ldots, M$, it runs in $\tilde{O}(\sqrt{M}n^{(\omega+3)/2})$ time where $\omega < 2.376$ is the exponent of matrix multiplication. As a by-product, our algorithm can be used to determine which vertices lie on cycles of length at most t in $\tilde{O}(Mn^{\omega}t)$ time. For the case of bounded real edge weights, a variant of our algorithm solves the problem up to an additive error of ϵ in $\tilde{O}(n^{(\omega+6)/3})$ time.

1 Introduction

Finding shortest cycles in a graph is among the most fundamental algorithmic graph problems. The problem has numerous applications and has been extensively studied.

Itai and Rodeh [6] presented an $O(nm)$-time algorithm that finds a shortest cycle (and hence, computes the *girth*) of a directed graph with n vertices and m edges. They also obtain an $O(n^{\omega})$ time algorithm for computing a shortest cycle of an unweighted undirected graph, where $\omega < 2.376$ is the matrix multiplication exponent [3]. For directed graphs their algorithm runs in $O(n^{\omega} \log n)$ time. The matrix multiplication based algorithms are faster than the $O(mn)$ algorithm when the graph is dense. Recently, Roditty and Vassilevska Williams [8] extended the result of Itai and Rodeh to the *integer weighted* setting. They design an $\tilde{O}(Mn^{\omega})$-time algorithm that finds a shortest cycle in a directed or undirected graph with integer edge weights in $\{1, \ldots, M\}$. Their algorithm can also handle weights in $\{-M, \ldots, M\}$ in the directed setting. If one settles for an approximation algorithm, then faster algorithms exist. For undirected graphs, Itai and Rodeh's algorithm can be made to run in $O(n^2)$-time, if one settles for a cycle whose length is at most one larger than the

girth. Alternatively, a shortest *even* length cycle can be found in $O(n^2)$-time using an algorithm of Yuster and Zwick [12]. Finally, Roditty and Tov [7] extended Itai and Rodeh's algorithm to undirected graphs with weights in $\{1, ..., M\}$ by showing that a cycle of length $g + W$ can be found in $\tilde{O}(n^2 \log M)$ time, where W is the maximum edge weight on a shortest cycle and g is the girth. Using this, they obtained an almost quadratic time 4/3-approximation for the girth. For fixed k, finding a cycle of length at most k (or, precisely k) if such a cycle exists, can be done in $\tilde{O}(n^\omega)$ time using the color-coding method of Alon et al. [1]. In fact, the color-coding method also finds in the same time, for each vertex, a cycle of length k containing it, if one exists. Their algorithm remains polynomial as long as $k = O(\log n)$. Unfortunately, the shortest cycle through a vertex may have larger length and hence, the color-coding method cannot solve the problem efficiently.

In this paper we present a new algorithm that finds, for each vertex in a graph, a shortest cycle containing it. Formally, given a graph $G = (V, E)$, the *all-nodes shortest cycle* (ANSC) problem seeks to find, for each $v \in V$, a cycle C_v containing v, with minimum weight among all cycles containing v. If v is not on any cycle, then $C_v = \emptyset$.

An $\tilde{O}(mn)$ time algorithm for ANSC is obtained as a by-product of the algorithm of Suurballe and Tarjan [11], as their algorithm can be used to find the shortest cycle through a given vertex in $O(m \log_{1+m/n} n)$ time. Observe that this algorithm runs in cubic time for dense graphs with $m = \Theta(n^2)$ edges, even if the graph is unweighted. ANSC can be easily solved for simple digraphs (directed graphs with no anti-parallel edges) by reducing it to the all-pairs shortest paths (APSP) problem. After running APSP, we can obtain the length of a shortest cycle through vertex v by just taking the minimum of $\delta(v, u) + w(u, v)$ where the minimum is taken over all incoming neighbors u of v. (Here $\delta(u, v)$ denotes the distance from u to v and $w(u, v)$ denotes the weight of the edge (u, v); for unweighted graphs we have $w(u, v) = 1$.) The presently fastest APSP algorithm for general weighted graphs is by Chan [2]. It runs in $O(n^3 \log^3 \log n / \log^2 n)$, and hence is only sub-cubic up to logarithmic factors. There are truly subcubic algorithms for APSP in unweighted directed graphs, and directed graphs with relatively small integer weights. The presently fastest such algorithm is by Zwick [13]. In the unweighted case, it runs in $O(n^{2.575})$ time, where the exponent is a function of rectangular matrix multiplication exponents. If $\omega = 2 + o(1)$, as may turn out to be the case, Zwick's algorithm runs in $\tilde{O}(n^{2.5})$ time. For graphs with integer edge weights of absolute value at most M, Zwick's algorithm runs in $\tilde{O}(M^{1/(4-\omega)} n^{2+1/(4-\omega)})$ (and slightly faster if rectangular matrix multiplication is used).

Unfortunately, the method of reducing ANSC to APSP fails when the digraph is not simple since a cycle may not use two anti-parallel edges. In particular, the method fails for undirected graphs. Our main result presents an algorithm that solves the ANSC problem for unweighted undirected graphs in truly sub-cubic time. It also applies to graphs with integer weights of value at most M.

Theorem 1.1 *There is a randomized algorithm that solves the ANSC problem in undirected graphs with weights in* $\{1, \ldots, M\}$. *The algorithm runs in* $\tilde{O}(\sqrt{M} n^{(\omega+3)/2})$ *time.*

With very high probability, the algorithm produces, for each $v \in V$, a shortest cycle through v of minimum weight.

As a by-product, our algorithm can be used to determine which vertices lie on cycles of length at most t in $\tilde{O}(Mn^{\omega}t)$ time. For the case of bounded real edge weights (or, more liberally, weights bounded by $n^{o(1)}$), a variant of our algorithm solves the ANSC problem up to an additive error of ϵ in $\tilde{O}(n^{(\omega+6)/3})$ time.

The rest of this paper is organized as follows. In Section 2 we prove Theorem 1.1 and its aforementioned variants. The final section contains some concluding remarks.

2 A shortest cycle for each vertex

Let $G = (V, E)$ be an undirected graph with n vertices and with integer edge weights in $\{1, \ldots, M\}$. It will be convenient to assume that G is 2-connected, as otherwise we can solve the problem in each 2-connected component separately. In particular, we may assume that each vertex is on some cycle, and that the length of the shortest cycle passing through v, denoted by $sc(v)$, is well-defined.

2.1 A shortest cycle through a given vertex

Let us first consider the much simpler task of finding a shortest cycle through a *given* vertex $s \in V$. We describe a simple randomized procedure that solves this problem.

A shortest cycle through s is composed of an edge (s, v), and a shortest path from s to v that avoids (s, v). To find the length of this s-to-v path we can run a single-source shortest path algorithm from s on the graph G without the edge (s, v). Therefore, if we knew which edge incident to s to remove, we could find the shortest cycle through s in $O(m + n \log n)$ time, using Dijkstra's algorithm [4, 5]. However, clearly we don't know which edge this is. Instead, we *guess* which edge incident to s we need to remove. Let $v_1, v_2, \ldots, v_{deg(s)}$ denote the neighbors of s. We start by removing each edge (s, v_j) independently with probability $\frac{1}{2}$ for every $j = 1, \ldots, deg(s)$. Then, we run the single-source shortest path algorithm from s on the resulting graph.

For each edge (s, v_j) that was previously removed, we estimate the length of the shortest cycle through (s, v_j) to be the weight of (s, v_j) plus the length of the shortest s-to-v_j path that we just computed. We pick the shortest cycle among all these as the shortest cycle through s. We now prove that with probability $\frac{1}{2}$ we found the actual shortest cycle through s.

Consider the actual shortest cycle through s; it includes s and exactly two of its neighbors, say v_a and v_b. With probability $\frac{1}{2}$ we have deleted (s, v_a) or (s, v_b) but not both. Therefore, with probability $\frac{1}{2}$ we found the actual shortest cycle through s in $O(m + n \log n)$ time. To amplify the probability to larger than $\frac{1}{2}$ we apply the same procedure constantly many times. So, with very high probability we can find the shortest cycle through s in $O(m + n \log n)$ time. We note that it is possible to derandomize the above procedure, resulting in an $O(\log n)$-factor increase in the runtime.

However, as some other ingredients of our algorithm are also randomized, we do not bother with derandomization.

Now, if we wish to compute a shortest cycle through a given vertex for *each* vertex in a set S, then, naively, the above algorithm can be applied to each vertex of S separately. The main idea in the proof of Theorem 1.1 is to somehow implement the above algorithm "all at once" thereby obtaining a significantly faster runtime.

2.2 Proof of the main result

For a vertex $v \in V$, let $c(v)$ denote the smallest number of edges in a shortest cycle realizing $sc(v)$. Let $C(v)$ be a set of $c(v)$ vertices on a shortest cycle through v. Although we will never explicitly compute $c(v)$ nor $C(v)$, it will be important for the *analysis* of our algorithm to fix one such $C(v)$ for each $v \in V$. For an integer $t \geq 3$ to be chosen later, let $V_t \subset V$ be those vertices for which $c(v) \leq t$.

Let $\mathcal{P}_t$ be the set of partitions of V into t parts, each consisting of n/t vertices (we ignore rounding issues as these have no effect on the asymptotic running times). Clearly, we can uniformly generate a random element of $\mathcal{P}_t$ in $O(n)$ time. Let $\mathcal{R}$ be a subset of $3 \log n$ elements of $\mathcal{P}_t$, each generated uniformly and independently (we do not care if partitions in $\mathcal{R}$ are repeating, although this happens with negligible probability).

Lemma 2.1 *With probability at least $1 - 1/n$, for each $v \in V_t$ there exists an element $P \in \mathcal{R}$ such that the part of P containing v does not contain any other element of $C(v)$.*

Proof: Consider an element P of $\mathcal{R}$. Recall that $P = P_1 \cup \cdots \cup P_t$ is a random partition of V into t parts of size n/t each. Let $v \in V_t$ and assume, without loss of generality, that $v \in P_1$. Suppose $C(v) = \{v, u_1, \ldots, u_r\}$ and note that $r < t$. The probability that $u_1 \notin P_1$ is precisely $1-(n/t-1)/(n-1) \geq 1-1/t$. Similarly, the probability that $u_j \notin P_1$ *given* that $\{u_1, \ldots, u_{j-1}\} \cap P_1 = \emptyset$ is $1 - (n/t - 1)/(n - j) \geq 1 - 1/t$. Hence,

$$\Pr[P_1 \cap C(v) = \{v\}] \geq \left(1 - \frac{1}{t}\right)^r > e^{-1} .$$

It follows that for a given $v \in V_t$, the probability that there exists an element $P \in \mathcal{R}$ such that the part of P containing v does not contain any other element of $C(v)$ is at least

$$1 - (1 - e^{-1})^{|\mathcal{R}|} = 1 - (1 - e^{-1})^{3 \log n} > 1 - \frac{1}{n^2} .$$

It follows from the union bound that with probability $1 - 1/n$, this holds for all $v \in V_t$. ■

The next lemma proves that a shortest cycle through each vertex of V_t can be computed, with very high probability, in $\tilde{O}(Mn^\omega t)$ time.

Lemma 2.2 *For $t \geq 3$, there is an algorithm that, with probability $1 - O(1/n)$, computes $sc(v)$ and a shortest cycle realizing $sc(v)$ for each vertex $v \in V_t$. For vertices in $V \setminus V_t$ the algorithm is not guaranteed to compute $sc(v)$. The runtime of the algorithm is $\tilde{O}(Mn^\omega t)$.*

Proof: We construct a random subset $\mathcal{R}$ of $3\log n$ elements of $\mathcal{P}_t$. Hence, $\mathcal{R} = \{P^{(1)}, \ldots, P^{(3\log n)}\}$, and $P^{(i)} = \{P_1^{(i)}, \ldots, P_t^{(i)}\}$ is a partition of V.

We do the following for each $P_j^{(i)}$ for $i = 1, \ldots 3\log n$ and $j = 1, \ldots, t$. For $k = 1, \ldots, 2\log n$, let $G_{i,j,k}$ be a spanning graph of G obtained by the following method. Each edge of G not incident with a vertex of $P_j^{(i)}$ also appears in $G_{i,j,k}$. Each edge of G incident with a vertex of $P_j^{(i)}$ is chosen to remain in $G_{i,j,k}$ with probability $1/2$. For a given i, j, all $2\log n$ graphs $G_{i,j,1}, \ldots, G_{i,j,2\log n}$ are generated independently.

We solve the APSP problem for each $G_{i,j,k}$ separately, and let $f_{i,j,k}(x, y)$ denote the computed distance between x and y in $G_{i,j,k}$. Since $G_{i,j,k}$ has n vertices, and each edge weight is in $\{1, \ldots, M\}$, the time to compute an APSP solution for $G_{i,j,k}$ is $\tilde{O}(Mn^\omega)$ using the algorithm of Shoshan and Zwick [9]. Hence, the overall runtime required for computing APSP solutions for all $G_{i,j,k}$ is

$$\tilde{O}(t|\mathcal{R}|2\log nMn^\omega) = \tilde{O}(Mn^\omega t)\ .$$

By lemma 2.1, with probability at least $1 - 1/n$, $\mathcal{R}$ is *good* in the sense that it has the property that for each $v \in V_t$ there exists $P_j^{(i)}$ such that $v \in P_j^{(i)}$, but no other vertex of $C(v)$ belongs to $P_j^{(i)}$. Let us assume, therefore, that R is good. Let $v \in V_t$ and let $P_j^{(i)}$ be such that $v \in P_j^{(i)}$ but no other vertex of $C(v)$ belongs to $P_j^{(i)}$. Let x, y be the two neighbors of v on the cycle $C(v)$. Notice that for each $k = 1, \ldots, 2\log n$ we have that with probability $1/2$, $G_{i,j,k}$ contains precisely one of the two edges (v, x) or (v, y). Suppose, without loss of generality, that $G_{i,j,k}$ contains (v, x) but does not contain (v, y). Then we must have $sc(v) = f_{i,j,k}(v, y) + w(v, y)$. Indeed, each edge on $C(v)$ other than (v, y) appears in $G_{i,j,k}$ because (v, x) appears, and all other edges of this cycle connect vertices not in $P_j^{(i)}$. Thus, $f_{i,j,k}(v, y) \leq sc(v) - w(v, y)$. It cannot be smaller, though, because otherwise the shortest cycle through v would have had weight smaller than $sc(v)$.

Our algorithm therefore proceeds as follows. For each $v \in V$ we do the following. For each $i = 1, \ldots, 3\log n$ let j be the unique part $P_j^{(i)}$ containing v. For each $k = 1, \ldots, 2\log n$ we consider all the values of the form $f_{i,j,k}(v, y) + w(v, y)$ taken over all neighbors y of v for which $(v, y) \notin G_{i,j,k}$. We take the minimum over all such values, and denote it by $f(v)$. Notice that $f(v)$, if finite, is indeed a length of a cycle containing v, since in the graph $G_{i,j,k}$ for which $f_{i,j,k}(v, y) + w(v, y) = f(v)$, the shortest path from v to y, together with the edge (v, y), form a cycle in G of weight $f(v)$. By the argument in the previous paragraph, if $\mathcal{R}$ is good then the probability that $f(v)$ is not the length of a shortest cycle through v is at most $(1/2)^{2\log n} = 1/n^2$. Thus, with probability at least $1 - 1/n - n/n^2 = 1 - 2/n$, we have that for each $v \in V_t$, the value $f(v)$ computed by the algorithm (and the cycle associated with it) is the length of a shortest cycle realizing $sc(v)$. For vertices not in V_t, the value $f(v)$ is either infinite, or is the length of a cycle containing v which is not necessarily a shortest cycle. ■

We are still left with the problem of correctly computing $sc(v)$ and a shortest path realizing it, for those vertices in $V \setminus V_t$. Moreover, we actually do not *know* the set V_t. Still, if we can find a way

to compute, with high probability, the values $sc(v)$ for the vertices in $V \setminus V_t$, then we can complement Lemma 2.2 and actually do not need to know V_t.

Suurballe and Tarjan [11], improving an earlier algorithm of Suurballe [10], described an algorithm that for a given a vertex s, builds a data structure that, given any vertex $v \in V$, can be used to construct two *edge-disjoint* paths from s to v of minimum total weight (their algorithm works for both directed and undirected graphs). The data structure contains an implicit representation of the n pairs of paths. The time required to construct it is $O(m \log_{1+m/n} n)$. For a given vertex v, the time necessary to explicitly construct the pair of paths connecting s and v is $O(1)$ per edge on the paths. We will make use of this algorithm in the following lemma.

Lemma 2.3 *For $t \geq 3$, there is an algorithm that, with probability $1 - O(1/n)$, computes $sc(v)$ and a shortest cycle realizing $sc(v)$ for each vertex $v \in V \setminus V_t$. For vertices in V_t the algorithm is not guaranteed to compute $sc(v)$. The runtime of the algorithm is $\tilde{O}(n^3/t)$.*

Proof: Let S be a random set set of $3n \log n/t$ vertices. We say that S is *good* if $S \cap (C(v) \setminus \{v\}) \neq \emptyset$ for each $v \in V \setminus V_t$. We claim that S is good with probability at least $1 - 1/n$. Indeed, let $v \in V \setminus V_t$. Since $C(v) > t$, the probability that $S \cap (C(v) \setminus \{v\}) = \emptyset$ is less than $(1 - t/n)^{|S|} < 1/n^2$. It follows that with probability at least $1 - 1/n$, for each $v \in V \setminus V_t$ some vertex of $C(v) \setminus \{v\}$ belongs to S.

We run the algorithm of Suurballe and Tarjan [11], once for every vertex $s \in S$ as a source. For a given $s \in S$, and for each $v \in V$, we can therefore compute two edge disjoint paths connecting s and v, denoted by $p_1(s, v)$ and $p_2(s, v)$, having minimum total weight, denoted by $g(s, v)$. The algorithm of [11] computes all these paths and values in $\tilde{O}(n^2)$ for a given $s \in S$, and hence in $\tilde{O}(n^3/t)$ for all $s \in S$. We also set $g(v) = \min_{s \in S} g(s, v)$.

We claim that if S is good then $g(v) = sc(v)$ for each $v \in V \setminus V_t$. Indeed, suppose $v \in V \setminus V_t$. Let $s \in S \cap C(v) \setminus \{v\}$. We claim that $p_1(s, v)$ and $p_2(s, v)$ are not only edge disjoint; they are also internally vertex-disjoint. Indeed, consider $C(v)$. It defines two internally vertex-disjoint paths from v to s, with total weight $sc(v)$. Thus, $g(s, v) \leq sc(v)$. On the other hand, if $p_1(s, v)$ and $p_2(s, v)$ were not internally vertex disjoint, then there is some vertex $w \neq s$ on both paths so that the part of p_1 connecting v and w and the part of p_2 connecting v and w together form a simple cycle of total length less than $g(s, v)$ and hence less than $sc(v)$, a contradiction. Now, since $p_1(s, v)$ and $p_2(s, v)$ are internally vertex-disjoint, they, together, form a simple cycle containing v and having weight $g(s, v)$. It therefore follows that we must have $g(s, v) = sc(v)$. For the same reason, we cannot have any vertex $s' \in S$ for which $g(s', v) < sc(v)$. It follows that $g(v) = sc(v)$, as required, and that the two paths realizing $g(v)$ form a simple cycle through v of minimum length. As S is good with probability at least $1 - 1/n$, the lemma follows. ■

Completing the proof of Theorem 1.1: We will set $t = n^{(3-\omega)/2} M^{-1/2}$. We then run the two algorithms of Lemma 2.2 and of Lemma 2.3. By our choice of t, the runtime of both algorithms is $\tilde{O}(\sqrt{M} n^{(\omega+3)/2})$. We therefore obtain, for each $v \in V$, the value $f(v)$ resulting from Lemma 2.2 and

the value $g(v)$ resulting from Lemma 2.3. We set $h(v) = \min\{g(v)\ ,\ f(v)\}$, and from Lemmas 2.2 and 2.3 we know that with probability $1 - O(1/n)$ we have that for each $v \in V$, $h(v) = sc(v)$. Notice that we can also generate a cycle having weight $h(v)$ for each $v \in V$ in the same $\tilde{O}(\sqrt{M}n^{(\omega+3)/2})$ time. ■

3 Extensions

The algorithm of Lemma 2.2 also applies to directed graphs. In this case, however, the runtime of the algorithm of Lemma 2.2 becomes slower, as it must now invoke applications of APSP in directed graphs, the fastest algorithm to date being Zwick's algorithm [13] running in $\tilde{O}(M^{1/(4-\omega)}n^{2+1/(4-\omega)})$ (in fact, a bit faster using fast rectangular matrix multiplication). Thus, the overall runtime of the algorithm of Lemma 2.2 becomes $\tilde{O}(M^{1/(4-\omega)}n^{2+1/(4-\omega)}t)$. This, however, can be improved by the following observation. Zwick's algorithm, actually computes shortest paths that use at most r edges in $\tilde{O}(Mn^{\omega}r^{3-\omega})$ time. When applying that algorithm of Lemma 2.2, we are only interested in shortest paths that use at most t edges (since we claim nothing on vertices in $V \setminus V_t$). It follows that the algorithm of Lemma 2.2 can be implemented in $\tilde{O}(Mn^{\omega}t^{4-\omega})$ time.

Given the target value t, deciding which vertices lie on simple cycles of length at most t, as well as computing $sc(v)$ for those vertices, can be done by just invoking the algorithm of Lemma 2.2, as, with very high probability, these are precisely the vertices for which $f(v) \leq t$. We therefore obtain the following corollary.

Corollary 3.1 *There is a randomized algorithm that, given a graph and an integer $t > 0$, determines the set of all vertices that lie on cycles of length at most t (and computes $sc(v)$ for those vertices). The algorithm runs in $\tilde{O}(Mn^{\omega}t)$ for undirected graphs and $\tilde{O}(\min\{Mn^{\omega}t^{4-\omega}\ ,\ M^{1/(4-\omega)}n^{2+1/(4-\omega)}\})$ for directed graphs.*

Notice that if both $t = n^{o(1)}$ and $M = n^{o(1)}$ the runtime becomes $\tilde{O}(n^{\omega})$ (in both the undirected and directed cases).

Suppose that the weights on the edges are positive reals and the largest weight is $W = n^{o(1)}$. Given an error requirement ϵ, it suffices for the algorithm of Lemma 2.2 to invoke an APSP algorithm that additively approximates paths with at most t edges up to an $O(\epsilon/(Wt))$ additive error. As the latter can be done in $\tilde{O}(n^{\omega}t)$ time using either the method from Section 2 or the stretch factor approximation algorithm from [13], the runtime of the algorithm of Lemma 2.2 becomes $\tilde{O}(n^{\omega}t^2)$. Since the algorithm of Surballe and Tarjan is capable of handling positive real weights, the runtime in Lemma 2.3 remains intact. By choosing $t = n^{(3-\omega)/3}$, the overall runtime is $\tilde{O}(n^{(\omega+6)/3})$. We therefore obtain the following corollary.

Corollary 3.2 *There is a randomized algorithm that solves the ANSC problem up to an ϵ-additive error in undirected graphs with positive real weights bounded by $n^{o(1)}$. The running time of the*

algorithm is $\tilde{O}(n^{(\omega+6)/3})$.

4 Concluding remarks

The algorithm of Theorem 1.1 is randomized. It would be interesting to obtain a deterministic version of it.

References

[1] N. Alon, R. Yuster, and U. Zwick, *Color coding*, Journal of the ACM 42 (1995), 844–856.

[2] T.M. Chan, *More algorithms for all-pairs shortest paths in weighted graphs*, Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC), ACM Press (2007), 590–598.

[3] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, Journal of Symbolic Computation 9 (1990), 251–280.

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

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

[6] A. Itai and M. Rodeh, *Finding a minimum circuit in a graph*, SIAM Journal on Computing 7 (1978), 413–423.

[7] L. Roditty and R. Tov, *Approximating the girth*, Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM-SIAM (2011), 1446-1454.

[8] L. Roditty and V. Vassilevska Williams, *Minimum weight cycles and triangles: equivalences and algorithms*, Proceedings of the 52nd IEEE conference on Foundations of Computer Science (FOCS), IEEE (2011), to appear.

[9] A. Shoshan and U. Zwick, *All pairs shortest paths in undirected graphs with integer weights*, Proceedings of the 40th IEEE conference on Foundations of Computer Science (FOCS), IEEE (1999), 605–614.

[10] J. W. Suurballe, *Disjoint Paths in Networks*, Networks 4 (1974), 125–145.

[11] J. W. Suurballe and R. E. Tarjan, *A quick method for finding shortest pairs of disjoint paths*, Networks 14 (1984), 325–336.

[12] R. Yuster and U. Zwick, *Finding even cycles even faster*, Siam Journal of Discrete Mathematics 10 (1997), 209–222.

[13] U. Zwick, *All-pairs shortest paths using bridging sets and rectangular matrix multiplication*, Journal of the ACM 49 (2002), 289–317.