Faster single-source shortest paths with negative real weights via proper hop distance*

Yufan Huang   Peter Jin   Kent Quanrud

Abstract

The textbook algorithm for single-source shortest paths with real-valued edge weights runs in $O(mn)$ time on a graph with m edges and n vertices. A recent breakthrough algorithm by Fineman [11] takes $\tilde{O}\big(mn^{8/9}\big)$ randomized time. We present an $\tilde{O}\big(mn^{4/5}\big)$ randomized time algorithm building on ideas from [11].

1 Introduction

Let $G = (V, E)$ be a directed graph with m edges, n vertices, and real-valued edge lengths $\ell : E \to \mathbb{R}$. For a fixed source vertex s, the single-source shortest path (SSSP) problem is to compute either a negative length cycle in G, or the shortest-path distance from s to every other vertex in G. The textbook $O(mn)$ time dynamic programming algorithm; discovered in the 1950's by Shimbel [19], Ford [12], Bellman [3], and Moore [17]; is a staple of introductory algorithms courses. Until very recently, it was also the fastest algorithm for this problem.

There are much faster algorithms for important special cases. When the edge weights are nonnegative, Dijkstra's algorithm takes $O(m + n \log n)$ time [9, 13]. There is also a long line of research when the edge weights are integer [14, 15, 18, 20, 8, 5, 2, 7, 4, 6], leading to a remarkable nearly linear running time in [4]. These algorithms all have at least a logarithmic dependence on the magnitude of the most negative edge weight.

The textbook bound of $O(mn)$ held its ground until a recent and exciting breakthrough by Fineman [11], who gave an $\tilde{O}\big(mn^{8/9}\big)$ randomized time algorithm in the Real RAM model. ($\tilde{O}(\cdots)$ hides $\log(n)$ factors.) Many of the interesting ideas in this work are detailed below in Section 2.

This work was inspired by [11]. Building on Fineman's ideas, we obtain:

THEOREM 1.1. *There is a (Las Vegas) randomized algorithm solving the SSSP problem for real-weighted graphs that runs in $O\big(mn^{4/5}\log^{2/5} n + n^{9/5}\log^{7/5} n\big)$ randomized time with high probability.*

The algorithm follows the same general framework laid out by [11], which eliminates negative edges incrementally along randomly sampled "independent sets" or "sandwiches" of negative edges. An interesting idea that we found useful was to focus on "proper" walks, where all the negative edges are required to be distinct, and on "proper hop distances", which consider proper walks with an exact number of negative edges (or "hops"). Morally, it is much harder for a random sample of negative edges to preserve a negative proper hop distance between two vertices, especially when the number of hops is large. This fact allows us to aggressively sample larger "sandwiches" for the same amount of effort. We explain in greater detail below.

2 Preliminaries

Let $G = (V, E)$ be a directed graph with m edges, n vertices, and real-valued edge lengths $\ell : E \to \mathbb{R}$. We let k denote the number of negative edges in G. The distance from u to v is defined as the infimum length over all walks from u to v. We let $d(u, v)$ denote the distance from u to v with respect to ℓ. More generally, for vertex sets S and T, we let $d(S, T)$ denote the minimum distance $d(s, t)$ over all $s \in S$ and $t \in T$.

Vertex potentials. Given vertex potentials $\varphi : V \to \mathbb{R}$, let $\ell_\varphi(e) \stackrel{\text{def}}{=} \ell(e) + \varphi(u) - \varphi(v)$ for an edge $e = (u, v)$. Johnson [16] observed that the potentials $\varphi(v) = d(V, v)$ give $\ell_\varphi(e) \geq 0$ for all e (assuming there are no negative-length cycles). We let G_φ denote the graph with edges reweighted by φ and d_φ denote distances with respect to ℓ_φ.

We say that potentials φ are *valid* if $\ell_\varphi(e) \geq 0$ for all e with $\ell(e) \geq 0$; i.e., φ does not introduce any new negative edges. Given two valid potentials φ_1 and φ_2, $\max\{\varphi_1, \varphi_2\}$ and $\min\{\varphi_1, \varphi_2\}$ are also valid potentials. If φ_1 is valid for G, and φ_2 is valid for G_{φ_1}, then $\varphi_1 + \varphi_2$ is valid for G.

We say that φ *neutralizes* a negative edge e if $\ell_\varphi(e) \geq 0$. This work, like other previous work since [15], iteratively computes and integrates valid potentials φ that neutralize more and more negative edges, until all edges are nonnegative.

Preprocessing and negative vertices. By standard techniques, we may assume the maximum in-degree and out-degree are both $O(m/n)$. We also preprocess the input graph G as follows [11]. For each vertex v, let $\mu_v = \min\{\ell(v, x) : (v, x) \in E\}$ be the minimum length of the edges leaving v. Consider the graph G' where we:

1. Replace each vertex v with two vertices v^-, v^+, and an arc (v^-, v^+) with length μ_v.
2. Replace each arc (u, v) with the arc (u^+, v^-) with length $\ell(u, v) - \mu_u$.

G' has similar size to G, and distances in G' give distances in G. Additionally, all the negative edges in G' have the form (v^-, v^+) for some vertex v. So there are at most $n/2$ negative edges, and each negative edge (u, v) is the unique outgoing edge of its tail u.

Henceforth we assume that G has at most $n/2$ negative edges, and that each negative edge (u, v) is the unique incoming edge of its head v and the unique outgoing edge of its tail u. We identify each negative edge (u, v) with its tail u, and call u a *negative vertex*. When there is no risk of confusion, we may reference a negative edge by its negative vertex and vice-versa. For example, we say we *neutralize a negative vertex* when we mean we neutralize its associated negative edge.

We let N denote the set of negative vertices in G. For a set of negative vertices S, we let G_S denote the subgraph obtained by restricting the set of negative edges to those corresponding to S.

When computing Johnson's potentials $\varphi(v) = d(V, v)$, any nontrivial shortest walk to v might as well start with a negative edge. Therefore $d(V, v) = \min\{0, d(N, v)\}$ for all v.

Hop distance. The number of *hops* in a walk is the number of negative edges along the walk, counted with repetition. An *r-hop walk* is a walk with at most r negative edges. For an integer r, and vertices $s, t \in V$, we let $d^r(s, t)$ denote the infimum length over all r-hop walks from s to t. Hop distances obey the recurrence

$$(2.1) \qquad d^{r+1}(s, t) = \min\left\{ d^r(s, t), \min_{u \in N} d^r(s, u) + d^1(u, t) \right\}.$$

For fixed s, if $d^{r+1}(s, v) = d^r(s, v)$ for all $v \in N$, then $d(s, v) = d^{r+1}(s, v)$ for all v. Also, for $r \in \mathbb{N}$ and any set of vertices S, the potentials $\varphi_1(v) = d^r(S, v)$ and $\varphi_2(v) = -d^r(v, S)$ are valid.

For fixed s, and $r \in \mathbb{N}$, one can compute the r-hop distance $d^r(s, v)$ for all $v \in V$ by a hybrid of Dijkstra's algorithm and Bellman-Ford-Moore-Shimbel in $O(rm + rn \log n)$ time [10, 4]. The algorithm can be inferred from (2.1): given $d^r(s, u)$ for all u, we can compute $\min_{u \in N} d^r(s, u) + d^1(u, t)$ for all t with a single call to Dijkstra's algorithm over an appropriate auxiliary graph. By maintaining parent pointers, we can also obtain the walks supporting the r-hop distances.

More generally, hop distance can be defined with respect to any set of vertices S that contains N, and the observations above would still hold. We need this generalization for the following reason. Our algorithm, following [11], reweights the graph along multiple valid potential functions in a single iteration. The initial potential functions in an iteration are not intended to neutralize any edges but rather massage the graph for subsequent steps. However, they may incidentally neutralize some negative edges, which can decrease the hop counts of walks and disrupt various assumptions about hop distances. To avoid this, like [11], we freeze the set of negative vertices at the beginning of each iteration. During an iteration, we continue to refer to a negative vertex as negative even if its associated edge becomes nonnegative. This ensures the identity $d_\varphi^h(u, v) = d^h(u, v) + \varphi(u) - \varphi(v)$ for any potential φ within an iteration.

Remote edges, betweenness, and sandwiches. We say that one vertex u can *negatively reach* another vertex v if there is a negative length walk from u to v. More specifically, we say that u can negatively reach v in h hops if there is a negative walk from u to v with at most h negative edges.

For a parameter $r \in \mathbb{N}$, and a set of negative vertices $U \subseteq V$, U is *r-remote* if U can reach at most n/r vertices via negative r-hop paths. They are important because they can be neutralized efficiently.

LEMMA 2.1. ([11, LEMMA 3.3]) *Let G have maximum in-degree and out-degree $O(m/n)$. There is an algorithm that, given a set of r-remote vertices U, returns either a set of potentials neutralizing U or a negative cycle in $O((r + |U|/r)(m + n \log n))$ time.*

(As stated, [11, Lemma 3.3] reports the existence of a negative cycle, but the algorithm can easily be amended to return the negative cycle. The same comment applies to Lemma 2.2 below.)

To generate r-remote sets, [11] introduced the notions of betweenness and negative sandwiches. For a parameter $r \in \mathbb{N}$, and $s, t \in V$, the *r-betweenness* is defined as the number of vertices v such that $d^r(s, v) + d^r(v, t) < 0$. [11] gave the following randomized procedure to reduce the betweenness of all pairs of vertices.

LEMMA 2.2. ([11, LEMMA 3.5]) *There is an $O\big(rbm \log n + rbn \log^2 n + b^2 n \log^2 n\big)$ time randomized algorithm that returns either a set of valid potentials φ or a negative cycle. With high probability, all pairs $(s, t) \in V \times V$ have r-betweenness (at most) n/b with respect to ℓ_φ.*

[11] defines a *negative sandwich* as a triple (s, U, t), where $s, t \in V$, and $U \subseteq N$, such that $d^1(s, u) < 0$ and $d^1(u, t) < 0$ for all $u \in U$. We generalize this as follows. For a parameter h, a *weak h-hop negative sandwich* is a triple (s, U, t) such that $d^h(s, u) + d^h(u, t) \leq 0$ for all $u \in U$. The following lemma, converting sandwiches into remote edges when the betweenness is low, generalizes [11, Corollary 3.9] from negative sandwiches to weak multi-hop negative sandwiches.

LEMMA 2.3. *Let (s, U, t) be a weak h-hop negative sandwich and let (s, t) have $(r + h)$-betweenness n/r. Then one can compute valid potentials φ making U r-remote in $O((r + h)(m + n \log n))$ time.*

Proof. Without loss of generality, we assume all the vertices are reachable from s. (Given potentials making U r-remote in the subgraph reachable from s, assign the remaining vertices the maximum of these potentials.) Define $\varphi(v) \stackrel{\text{def}}{=} \max\big\{d^{r+h}(s, v), -d^{r+h}(v, t)\big\}$ for $v \in V$. For all $u \in U$,

$$-d^{r+h}(u, t) \geq -d^h(u, t) \stackrel{(a)}{\geq} d^h(s, u) \geq d^{r+h}(s, u),$$

where (a) is because $d^h(s, u) + d^h(u, t) \leq 0$. Thus $\varphi(u) = -d^{r+h}(u, t) \geq d^h(s, u)$ for all $u \in U$.

For $v \in V$ with $d^{r+h}(s, v) + d^{r+h}(v, t) \geq 0$, since $d^{r+h}(s, v) \geq -d^{r+h}(v, t)$, we have

$$\varphi(v) = d^{r+h}(s, v).$$

Now, let $u \in U$ and $v \in V$ with $d^{r+h}(s, v) + d^{r+h}(v, t) \geq 0$. We have

$$d_\varphi^r(u, v) = d^r(u, v) + \varphi(u) - \varphi(v) \geq d^r(u, v) + d^h(s, u) - d^{r+h}(s, v) \geq 0$$

by the triangle inequality.

Thus a vertex v is negatively reachable in r steps from U only if $d^{r+h}(s, v) + d^{r+h}(v, t) < 0$. There are only n/r such vertices, making U r-remote. As for the running time, it takes $O((r + h)(m + n \log n))$ time to compute $\varphi(v)$ for all v. $\qquad \square$

Sketch of [11]. To help motivate the preliminaries above, and to place ensuing ideas in their proper context, we briefly sketch the existing $\tilde{O}\big(mn^{8/9}\big)$ randomized time algorithm of [11].

Call a set of negative vertices U *independent* if $d^1(U, x) \geq 0$ for all $x \in U$. If U is independent, then in the graph restricting the negative vertices to U, we have $d(V, x) = d^1(V, x)$ for all x. Thus an independent set can be neutralized by Johnson's technique in nearly linear time.

Suppose there are currently k negative vertices and edges in G. [11] gives a randomized algorithm that produces either an independent set or a negative sandwich of size $\Omega\big(k^{1/3}\big)$. The independent set can be neutralized

directly in nearly linear time. To neutralize the negative sandwich with Lemma 2.1, we need the endpoints of the sandwich to have low betweenness. Thus [11] first spends $\tilde{O}\left(mk^{2/9}\right)$ time to decrease the $k^{1/9}$-betweenness to $n/k^{1/9}$ for all pairs of vertices, before sampling the independent set or negative sandwich. Then, if the sample returns a negative sandwich, the negative sandwich is neutralized in $\tilde{O}\left(mk^{2/9}\right)$ time. One way or another, it takes $\tilde{O}\left(mk^{2/9}\right)$ time to neutralize $k^{1/3}$ negative edges. It takes $\tilde{O}\left(mn^{8/9}\right)$ randomized time overall to repeat these steps until there are no negative edges remaining.

3 Proper multi-hop walks

Let $h \in \mathbb{N}$ be a parameter to be determined. Consider the following stricter notion of hops and hop distance. We define a *proper h-hop walk* as a walk with exactly h negative edges and where all negative vertices are distinct. For vertices s, t, let $\hat{d}^h(s, t)$ denote the infimum length over all proper h-hop walks from s to t. We call $\hat{d}^h(s, t)$ the *proper h-hop distance* from s to t.

Computing proper hop distances is tricky because negative vertices cannot repeat. Even if there are no negative cycles, and all distances are attained by paths, standard algorithms will still want to reuse negative edges in order to meet the hop requirement. For small values of h, single-source proper h-hop distances can be computed with high probability in $\tilde{O}\left(2^{O(h)}m\right)$ randomized time using color-coding techniques [1]. Fortunately, we can sidestep computing proper h-hop distances altogether, and take h to be much larger, with the following lemma.

LEMMA 3.1. *There is an $O(h(m + n \log n))$-time algorithm that, given a set of negative vertices S, returns either a negative cycle, a pair $s, t \in S$ with $\hat{d}^h(s, t) < 0$, or the distances $d(V, t)$ in G_S for all $t \in V$.*

Proof. Consider the subgraph G_S. We first compute $d^{h-1}(S, t)$ and $d^h(S, t)$ (and the supporting walks) for all $t \in V$. If $d^h(S, t) = d^{h-1}(S, t)$ for all $t \in S$, then $d(V, t) = \min\{0, d(S, t)\} = \min\{0, d^h(S, t)\}$ for all $t \in V$.

Otherwise, $d^h(S, t) < d^{h-1}(S, t) \leq 0$ for some $t \in S$. Consider the h-hop walk attaining $d^h(S, t)$. We know this walk has exactly h negative edges because it has length less than $d^{h-1}(S, t)$. If some negative vertex repeats, then the cycle along the walk between occurrences must be a negative cycle, since otherwise removing the cycle would give an $(h-1)$-hop walk to t with length less than $d^{h-1}(S, t)$. If the negative vertices are distinct, then we have a proper h-hop walk to t with negative length, as desired. $\quad\square$

4 Sampling weak negative sandwiches

Recall that weak negative sandwiches can be efficiently neutralized by the techniques in Section 2. Similar to [11], we try to find large sandwiches quickly by random sampling. We leverage proper h-hop distances to increase the size of the sandwich by a factor of $\Omega(\sqrt{h})$.

LEMMA 4.1. *For $h = \Omega(\log n)$, there is a randomized algorithm that, in $O(h(m + n \log n))$ randomized time, returns either a negative cycle, a weak h-hop sandwich (s, U, t), or a set of negative vertices S and the distances $d(V, v)$ for all vertices v in the subgraph G_S. With high probability, we have $|U|, |S| \geq \Omega\left(\sqrt{hk}\right)$ (when they are returned by the algorithm).*

Proof. Let $q = 2\sqrt{k/h}$. Let $S \subseteq N$ randomly sample each negative vertex independently with probability $1/q$. S has at least $k/2q = \sqrt{kh}/4$ negative vertices with high probability by the multiplicative Chernoff bound. Applying Lemma 3.1 to S, we obtain either (a) a negative cycle, (b) the distances $d(V, t)$ for all vertices t in G_S, or (c) a pair of $s, t \in S$ with $\hat{d}^h(s, t) < 0$. The negative cycle and the distances are returned immediately. In the third case, we return the sandwich $(s, B_{s,t}, t)$, where $B_{s,t} \overset{\text{def}}{=} \{u \in N : d^h(s, u) + d^h(u, t) < 0\}$, in $O(h(m + n \log n))$ time.

We say that an ordered pair (s, t) is *sampled by S* if $s \in S$, $t \in S$, and $\hat{d}^h(s, t) < 0$ in G_S. Call (s, t) *weak* if $|B_{s,t}| \leq \sqrt{kh}$, and *strong* otherwise. We claim that with high probability no weak pairs are sampled by S.

This is where proper hop distance comes to the fore: any proper h-hop walk from s to t with negative length must include at least $h - 1$ vertices from $B_{s,t} \setminus \{s, t\}$. For fixed s, t, $|B_{s,t} \cap S|$ is a sum of independent $\{0, 1\}$-random variables. If (s, t) is weak, then this sum has mean at most $h/2$. Since $h = \Omega(\log n)$, by the multiplicative Chernoff bound, $|B_{s,t} \cap S| \leq h$, hence (s, t) is not sampled, with high probability.

By the union bound, S samples at least $\sqrt{kh}/4$ negative vertices and no weak pairs with high probability. Then $|S|$ and $|B_{s,t}|$ (when available) both have the desired size. $\quad\square$

5 Putting it all together

We now present the overall algorithm and analysis, combining ingredients from Section 2 with the new sampling techniques from Section 4.

THEOREM 5.1. *There is a (Las Vegas) randomized algorithm solving the SSSP problem for real-weighted graphs that runs in $O\left(mn^{4/5}\log^{2/5}n + n^{9/5}\log^{7/5}n\right)$ randomized time with high probability.*

Proof. The algorithm iteratively tries to compute valid potentials φ that neutralizes some of the remaining negative edges, until no negative edges remain. We describe a single iteration.

Suppose there are k negative edges. We assume that $k = \Omega\left(\log^7 n\right)$, since otherwise we can use Johnson's technique with k-hop distances in $O(k(m + n\log n))$ time. Let $h = k^{1/5}/\log^{2/5}n = \Omega(\log n)$. By Lemma 2.2, in $O\left(h^2(m\log n + n\log^2 n)\right)$ time, we compute either a negative cycle (and halt) or compute valid potentials φ_1 that, with high probability, reduce the $O(h)$-betweenness of all pairs of vertices to n/h. Henceforth we work in G_{φ_1}.

Next we invoke Lemma 4.1. If it returns a negative cycle then we halt. If it returns a set of negative vertices S, and all the Johnson potentials $\varphi_2(v) = d(V, v)$ in G_S, then we re-weight the graph with φ_2 and neutralize S.

Otherwise we obtain a weak h-hop negative sandwich (s, U, t). If the betweenness-reduction succeeded, then by Lemma 2.3, we either return a negative cycle or compute potentials φ_2 that make U h-remote in $O(h(m + n\log n))$ time. Note that we can test if U is indeed h-remote in the same amount of time. If not, then the betweenness-reduction failed, and we restart the iteration. Otherwise, by Lemma 2.1, we either find a negative cycle (and halt) or compute potentials φ_3 neutralizing U (in $G_{\varphi_1+\varphi_2}$) in $O((h + |U|/h)(m + n\log n))$ time.

Altogether, with high probability of success, and in $O\left(\left(h^2\log n + \ell/h\right)(m + n\log n)\right)$ randomized time, we either neutralize $\ell = \Omega\left(\sqrt{kh}\right)$ negative edges, or return a negative cycle.

Starting from at most n negative vertices, it takes $O\left(\sqrt{n/h}\log n\right)$ successful iterations to reduce the number of negative vertices to $O\left(\log^7 n\right)$. By the union bound, all the iterations succeed with high probability.

The running time is dominated by the time spent to reduce the number of negative vertices by half. Thus the overall running time is $O\left((n^{1/2}h^{3/2}\log n + n/h)(m + n\log n)\right) = O\left(mn^{4/5}\log^{2/5}n + n^{9/5}\log^{7/5}n\right)$ with high probability. $\qquad\square$

References

[1] Noga Alon, Raphael Yuster, and Uri Zwick. "Color-Coding". In: *J. ACM* 42.4 (1995), pp. 844–856.

[2] Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. "Circulation Control for Faster Minimum Cost Flow in Unit-Capacity Graphs". In: *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*. 2020, pp. 93–104.

[3] Richard Bellman. "On A Routing Problem". In: *Quarterly of Applied Mathematics* 16.1 (1958), pp. 87–90.

[4] Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. "Negative-Weight Single-Source Shortest Paths in Near-linear Time". In: *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*. 2022, pp. 600–611.

[5] Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. "Bipartite Matching in Nearly-linear Time on Moderately Dense Graphs". In: *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*. 2020, pp. 919–930.

[6] Karl Bringmann, Alejandro Cassis, and Nick Fischer. "Negative-Weight Single-Source Shortest Paths in Near-Linear Time: Now Faster!" In: *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*. 2023, pp. 515–538.

[7] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. "Maximum Flow and Minimum-Cost Flow in Almost-Linear Time". In: *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*. 2022, pp. 612–623.

[8] Michael B. Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. "Negative-Weight Shortest Paths and Unit Capacity Minimum Cost Flow in $\tilde{O}(m^{10/7} \log W)$ Time (Extended Abstract)". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19.* 2017, pp. 752–771.

[9] Edsger W. Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1 (1959), pp. 269–271.

[10] Yefim Dinitz and Rotem Itzhak. "Hybrid Bellman-Ford-Dijkstra algorithm". In: *J. Discrete Algorithms* 42 (2017), pp. 35–44.

[11] Jeremy T. Fineman. "Single-Source Shortest Paths with Negative Real Weights in $\tilde{O}(mn^{8/9})$ Time". In: *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024.* 2024, pp. 3–14.

[12] Lester R. Ford. *Network Flow Theory.* Santa Monica, CA: RAND Corporation, 1956.

[13] Michael L. Fredman and Robert Endre Tarjan. "Fibonacci heaps and their uses in improved network optimization algorithms". In: *J. ACM* 34.3 (1987), pp. 596–615.

[14] Harold N. Gabow and Robert Endre Tarjan. "Faster Scaling Algorithms for Network Problems". In: *SIAM J. Comput.* 18.5 (1989), pp. 1013–1036.

[15] Andrew V. Goldberg. "Scaling Algorithms for the Shortest Paths Problem". In: *SIAM J. Comput.* 24.3 (1995), pp. 494–504.

[16] Donald B. Johnson. "Efficient Algorithms for Shortest Paths in Sparse Networks". In: *J. ACM* 24.1 (1977), pp. 1–13.

[17] Edward F. Moore. "The shortest path through a maze". In: *Proceedings of an International Symposium on the Theory of Switching 1957, Part II.* Cambridge, Massachusetts: Harvard University Press, 1959.

[18] Piotr Sankowski. "Shortest Paths in Matrix Multiplication Time". In: *Algorithms - ESA 2005, 13th Annual European Symposium, Palma de Mallorca, Spain, October 3-6, 2005, Proceedings.* Vol. 3669. Lecture Notes in Computer Science. Springer, 2005, pp. 770–778.

[19] Alfonso Shimbel. "Structure in communication nets". In: *Proceedings of the Symposium on Information Networks.* New York, New York: Polytechnic Press of the Polytechnic Institute of Brooklyn, 1955, pp. 199–203.

[20] Raphael Yuster and Uri Zwick. "Answering distance queries in directed graphs using fast matrix multiplication". In: *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings.* IEEE Computer Society, 2005, pp. 389–396.