Exact and approximate distances in graphs – a survey

Uri Zwick *

School of Computer Science
Tel Aviv University, Tel Aviv 69978, Israel
zwick@cs.tau.ac.il
http://www.cs.tau.ac.il/~zwick/

Abstract. We survey recent and not so recent results related to the computation of exact and approximate distances, and corresponding shortest, or almost shortest, paths in graphs. We consider many different settings and models and try to identify some remaining open problems.

1 Introduction

The problem of finding distances and shortest paths in graphs is one of the most basic, and most studied, problems in algorithmic graph theory. A great variety of intricate and elegant algorithms were developed for various versions of this problem. Nevertheless, some basic problems in this area of research are still open. In this short survey, I will try to outline the main results obtained, and mention some of the remaining open problems.

The input to all versions of the problem is a graph $G = (V, E)$. The graph G may be *directed* or *undirected*, and it may be *weighted* or *unweighted*. If the graph is weighted, then each edge $e \in E$ has a weight, or length, $w(e)$ attached to it. The edge weights are either arbitrary *real* numbers, or they may be *integers*. In either case, the weights may be *nonnegative*, or may allowed to be *negative*.

We may be interested in the distances and shortest paths from a single source vertex s to all other vertices of the graph, this is known as the *Single-Source Shortest Paths (SSSP)* problem, or we may be interested in the distances and shortest paths between all pairs of vertices in the graph, this is known as the *All-Pairs Shortest Paths (APSP)* problem. We may insist on getting *exact* distances and genuine shortest paths, or we may be willing to settle for *approximate* distances and almost shortest paths. The errors in the approximate distances that we are willing to accept may be of an *additive* or *multiplicative* nature.

If we insist on explicitly obtaining the distances between any pair of vertices in the graph, then the size of the output is $\Omega(n^2)$, where n is the number of vertices of the graph. Perhaps this is not what we had in mind. We may be

* Work supported in part by **The Israel Science Foundation** founded by The Israel Academy of Sciences and Humanities.

interested, for example, only in a concise *implicit* approximation to all the distances. This may be achieved, for example, by finding a sparse subgraph that approximates all the distances in G. Such a subgraph is called a *spanner*.

Finally, perhaps the implicit approximation of all the distances offered by spanners is not enough. We may want a concise representation of approximate distances, together with quick means of extracting these approximations when we need them. This leads us to the study of *approximate distance oracles*.

This summarizes the different problems considered in this survey. We still need to specify, however, the computational models used. We use two different variants of the unit cost *Random Access Machine* (RAM) model (see [1]). When the edge weights are real numbers, the only operations we allow on the edge weights, and the numbers derived from them, are *addition* and *comparison*. These operations are assumed to take $O(1)$ time. No other operations on real numbers are allowed. We call this the *addition-comparison model*. It is reminiscent of the algebraic computation tree model (see, e.g., [10]), though we are counting all operations, not only those that manipulate weights, and want a single concise program that works for any input size. When the edge weights are integral, we adopt the *word RAM* model that opens the way for more varied algorithmic techniques. In this model, each word of memory is assumed to be w-bit wide, capable of holding an integer in the range $\{-2^{w-1}, \ldots, 2^{w-1}-1\}$. We assume that every *distance* in the graph fits into one machine word. We also assume that $w \geq \log n$, so that, for example, the name of vertex can be stored in a single machine word. Other than that, no assumptions are made regarding the relation between n and w. We are allowed to perform additions, subtractions, comparisons, *shifts*, and various logical *bit operations*, on machine words. Each such operation takes only $O(1)$ time. (Surprisingly, shifts, ANDs, XORs, and the other such operations, that seem to have little to do with the problem of computing shortest paths, do speed up algorithms for the problem, though only by sub-logarithmic factors.) In some cases, we allow operations like *multiplication*, but generally, we try to avoid such non-AC^0 operations. See [39] for a further discussion of this model.

Most of the algorithmic techniques used by the algorithms considered are *combinatorial*. There are, however, some relations between distance problems and *matrix multiplication*. Thus, some of the algorithms considered, mostly for the APSP problem, rely on fast matrix multiplication algorithms (see, e.g., [18]). Some of the algorithms considered are *deterministic* while others are *randomized*.

There are many more interesting variants of problems related to distances and shortest paths that are *not* considered in this short survey. These include: Algorithms for restricted families of graphs, such as planar graphs (see, e.g., [43, 72]); Parallel algorithms (see, e.g., [49, 13, 15]); Algorithms for dynamic versions of the problems (see, e.g., [48, 21]); Routing problems (see, e.g., [20, 57, 74]); Geometrical problems involving distances (see, e.g., [28, 54, 55]); and many more.

Also, this short survey adopts a *theoretical* point of view. Problems involving distances and shortest paths in graphs are not only great mathematical problems, but are also very practical problems encountered in everyday life. Thus, there is great interest in developing algorithms for these problems that work well in practice. This, however, is a topic for a different survey that I hope someone else would write. For some discussion of practical issues see, e.g., [11, 12].

2 Basic definitions

Let $G = (V, E)$ be a graph. We let $|V| = n$ and $|E| = m$. We always assume that $m \geq n$. The *distance* $\delta(u, v)$ from u to v in the graph is the smallest length of a path from u to v in the graph, where the length of a path is the sum of the weights of the edges along it. If the graph is unweighted then the weight of each edge is taken to be 1. If the graph is directed, then the paths considered should be directed. If there is no (directed) path from u to v in the graph, we define $\delta(u, v) = +\infty$. If all the edge weights are nonnegative, then all distances are well defined. If the graph contains (directed) cycles of negative weight, and there is a path from u to v that passes through such a negative cycle, we let $\delta(u, v) = -\infty$.

Shortest paths from a source vertex s to all other vertices of the graph can be compactly represented using a *tree of shortest paths*. This is a tree, rooted at s, that spans all the vertices reachable from s in the graph, such that for every vertex v reachable from s in the graph, the unique path in the tree from s to v is a shortest path from s to v in the graph. Almost all the algorithms we discuss return such a tree (or a tree of almost shortest paths), or some similar representation. In most cases, producing such a tree is straightforward. In other cases, doing so without substantially increasing the running time of the algorithm is a non-trivial task. Due to lack of space, we concentrate here on the computation of exact or approximate distances, and only briefly mention issues related to the generation of a representation of the corresponding paths.

3 Single-source shortest paths

We begin with the single-source shortest paths problem. The input is a graph $G = (V, E)$ and a *source* $s \in V$. The goal is to compute all the distances $\delta(s, v)$, for $v \in V$, and construct a corresponding shortest paths tree. The following subsections consider various versions of this problem.

3.1 Nonnegative real edge weights

If the input graph $G = (V, E)$ is unweighted, then the problem is easily solved in $O(m)$ time using *Breadth First Search (BFS)* (see, e.g., [19]). We suppose, therefore, that each edge $e \in E$ has a nonnegative real edge weight $w(e) \geq 0$ associated with it. The problem can then be solved using the classical *Dijkstra's algorithm* [22]. For each vertex of G, we hold a *tentative distance* $d(v)$. Initially $d(s) = 0$, and $d(v) = +\infty$, for every $v \in V - \{s\}$. We also keep a set T of *unsettled vertices*. Initially $T = V$. At each stage we choose an unsettled vertex u with minimum tentative distance, make it settled, and explore the edges emanating from it. If $(u,v) \in E$, and v is still unsettled, we update the tentative distance of v as follows: $d(v) \leftarrow \min\{d(v), d(u) + w(u,v)\}$. This goes on until all vertices are settled. It is not difficult to prove that when a vertex u becomes settled we have $d(u) = \delta(s,u)$, and that $d(u)$ would not change again.

An efficient implementation of Dijkstra's algorithm uses a *priority queue* to hold the unsettled vertices. The *key* associated with each unsettled vertex is its tentative distance. Vertices are inserted into the priority queue using *insert* operations. An unsettled vertex with minimum tentative distance is obtained using an *extract-min* operation. Tentative distances are updated using *decrease-key* operations.

A simple heap-based priority queue can perform insertions, extract-min and decrease-key operations in $O(\log n)$ worst case time per operation. This gives immediately an $O(m \log n)$ time SSSP algorithm. *Fibonacci heaps* of Fredman and Tarjan [30] require only $O(1)$ *amortized* time per insert and decrease-key operation, and $O(\log n)$ *amortized* time per extract-min operation. This gives an $O(m + n \log n)$ time SSSP algorithm. (*Relaxed heaps* of Driscoll *et al.* [24] may also be used to obtain this result. They require $O(1)$ worst case (not amortized) time for per decrease-key operation, and $O(\log n)$ worst case time per extract-min operation.) The only operations performed by these algorithms on edge weights are additions and comparisons. Furthermore, every sum of weights computed by these algorithms is the length of a path in the graph. The $O(m + n \log n)$ time algorithm is the fastest known algorithm in this model.

Dijkstra's algorithm produces the distances $\delta(s,v)$, for $v \in V$, in *sorted order*. It is clear that this requires, in the worst case, $\Omega(n \log n)$ comparisons. (To sort n elements, form a star with n leaves and attach the elements to be sorted as weights to the edges.) However, the definition of the SSSP problem does not require the distances to be returned in sorted order. This leads us to our first open problem: Is there an algorithm for the SSSP problem in the addition-comparison model that beats the information theoretic $\Omega(n \log n)$ lower bound for sorting? Is there such an algorithm in the algebraic computation tree model?

3.2 Nonnegative integer edge weights – Directed graphs

We consider again the single-source shortest paths problem with nonnegative edge weights. This time we assume, however, that the edge weights are integral and that we can do more than just add and compare weights. This leads to improved running times. (The improvements obtained are sub-logarithmic, as the running time of Dijkstra's algorithm in the addition-comparison model is already almost linear.)

The fact that the edge weights are now integral opens up new possibilities. Some of the techniques that can be applied are *scaling*, *bucketing*, *hashing*, *bit-level parallelism*, and more. It is also possible to tabulate solutions of small subproblems. The description of these techniques is beyond the scope of this survey. We merely try to state the currently best available results.

Most of the improved results for the SSSP problem in the word RAM model are obtained by constructing improved priority queues. Some of the pioneering results here were obtained by van Emde Boas *et al.* [75, 76] and Fredman and Willard [31, 32]. It is enough, in fact, to construct *monotone* priority queues, i.e., priority queues that are only required to support sequences of operations in which the value of the minimum key never decreases.

Thorup [71] describes a priority queue with $O(\log\log n)$ expected time per operation. This gives immediately an $O(m \log\log n)$ expected time algorithm for the SSSP problem. (Randomization is needed here, and in most other algorithms, to reduce the work space needed to linear.) He also shows that the SSSP problem is not harder than the problem of sorting the m edge weights. Han [42] describes a deterministic sorting algorithm that runs in $O(n \log\log n \log\log\log n)$ time. This gives a deterministic, $O(m \log\log n \log\log\log n)$ time, linear space algorithm for the SSSP problem. Han's algorithm uses multiplication. Thorup [68] describes a deterministic, $O(n(\log\log n)^2)$ time, linear space sorting algorithm that does not use multiplication, yielding a corresponding SSSP algorithm.

Improved results for graphs that are not too sparse may be obtained by constructing (monotone) priority queues with constant (amortized) time per decrease-key operation. Thorup [71] uses this approach to obtain an $O(m + (n \log n)/w^{1/2-\epsilon})$ expected time algorithm, for any $\epsilon > 0$. (Recall that w is the width of the machine word.) Raman [61], building on results of Ahuja *et al.* [2] and Cherkassky *et al.* [12], obtains an $O(m + nw^{1/4+\epsilon})$ expected time algorithm, for any $\epsilon > 0$. Note that the first algorithm is fast when w is large, while the second algorithm is fast when w is small. By combining these algorithms, Raman [61] obtains an $O(m + n(\log n)^{1/3+\epsilon})$ expected time algorithm, for any $\epsilon > 0$. Building on his results from [60], he also obtains deterministic algorithms with running times of $O(m + n(w \log w)^{1/3})$ and $O(m + n(\log n \log\log n)^{1/2})$.

It is interesting to note that w may be replaced in the running times above by $\log C$, where C is the largest edge weight in the graph.

Finally, Hagerup [40], extending a technique of Thorup [69] for undirected graphs, obtains a deterministic $O(m \log w)$ time algorithm.

Is there a linear time algorithm for the directed SSSP problem in the word RAM model? Note that a linear time sorting algorithm would give an affirmative answer to this question. But, the SSSP problem may be easier than sorting.

3.3 Nonnegative integer edge weights – undirected graphs

All the improved algorithms mentioned above (with one exception) are 'just' intricate implementations of Dijkstra's algorithm. They produce, therefore, a sorted list of the distances and do not avoid, therefore, the sorting bottleneck.

In a sharp contract, Thorup [69, 70] developed recently an elegant algorithm that avoids the rigid settling order of Dijkstra's algorithm. Thorup's algorithm bypasses the sorting bottleneck and runs in optimal $O(m)$ time! His algorithm works, however, only on *undirected* graphs. It remains an open problem whether extensions of his ideas could could be used to obtain a similar result for directed graphs. (Some results along these lines were obtained by Hagerup [40].)

3.4 Positive and negative real edge weights

We now allow, for the first time, negative edge weights. The gap, here, between the best upper bound and the obvious lower bound is much wider.

Dijkstra's algorithm breaks down in the presence of negative edge weights. The best algorithm known for the problem in the addition-comparison model is the simple $O(mn)$ time Bellman-Ford algorithm (see, e.g., [19]): Start again with $d(s) = 0$ and $d(v) = +\infty$, for $v \in V - \{s\}$. Then, perform the following n times: For every edge $(u, v) \in E$, let $d(v) \leftarrow \min\{d(v), d(u) + w(u, v)\}$. If any of the tentative distances change during the last iteration, then the graph contains a negative cycle. Otherwise, $d(v) = \delta(s, v)$, for every $v \in V$.

The problem of deciding whether a graph contains a negative cycles is a special case of the problem for finding a *minimum mean weight* cycle in a graph. Karp [47] gives an $O(mn)$ time algorithm for this problem.

Is there a $o(mn)$ time algorithm for the single-source shortest paths problem with positive and negative weights in the addition-comparison model?

3.5 Positive and negative integer edge weights

Goldberg [38], improving results of Gabow [33] and of Gabow and Tarjan [34], uses scaling to obtain an $O(mn^{1/2} \log N)$ time algorithm for this version of the problem, where N is the absolute value of the smallest edge weight. (It is assumed that $N \geq 2$.) Goldberg's algorithm, like most algorithms dealing with negative edge weights, uses *potentials* (see Section 4.2). No progress on the problem was made in recent years. Is there a better algorithm?

4 All-pairs shortest paths – exact results

We now move to consider the all-pairs shortest paths problem. The input is again a graph $G = (V, E)$. The required output is a matrix holding all the distances $\delta(u, v)$, for $u, v \in V$, and some concise representation of all shortest paths, possibly a shortest path tree rooted at each vertex.

4.1 Nonnegative real edge weights

If all the edge weights are nonnegative, we can simply run Dijkstra's algorithm independently from each vertex. The running time would be $O(mn + n^2 \log n)$. Karger, Koller and Phillips [46] and McGeoch [52] note that by orchestrating the operation of these n Dijkstra's processes, some unnecessary operations may saved. This leads to an $O(m^*n + n^2 \log n)$ time algorithm for the problem, where m^* is the number of *essential* edges in G, i.e., the number of edges that actually participate in shortest paths. In the worst case, however, $m^* = m$.

Karger *et al.* [46] also introduce the notion of *path-forming* algorithms. These are algorithms that work in the addition-comparison model, with the additional requirement that any sum of weights computed by the algorithm is the length of some path in the graph. (All the addition-comparison algorithms mentioned so far satisfy this requirement.) They show that any path-forming algorithm for the APSP problem must perform, in the worst case, $\Omega(mn)$ operations.

4.2 Positive and Negative real edge weights

When some of the edge weights are negative, Dijkstra's algorithm cannot be used directly. However, Johnson [45] observed that if there are no negative weight cycles, then new nonnegative edge weights that preserve shortest paths can be computed in $O(mn)$ time. The idea is very simple. Assign each vertex $v \in V$ a *potential* $p(v)$. Define new edge weights as follows $w_p(u,v) = w(u,v)+p(u)-p(v)$, for every $(u,v) \in E$. It is easy to verify that the new distances satisfy $\delta_p(u,v) = \delta(u,v) + p(u) - p(v)$, for every $u, v \in V$. (The potentials along any path from u to v cancel out, except those of u and v.) Thus, the shortest paths with respect to the new edge weights are also shortest paths with respect to the original edge weights, and the original distances can be easily extracted. Now, add to G a new vertex s, and add zero weight edges from it to all other vertices of the graph. Let $p(v) = \delta(s,v)$. If there are no negative weight cycles in the graph then these distances are well defined and they could be found in $O(mn)$ time using the Bellman-Ford algorithm. The *triangle inequality* immediately implies that $w_p(u,v) \geq 0$, for every $u, v \in V$. Now we can run Dijkstra's algorithm from each vertex with the new nonnegative weights. The total running time is again $O(mn + n^2 \log n)$.

As m may be as high as $\Omega(n^2)$, the running time of Johnson's algorithm may be as high as $\Omega(n^3)$. A running time of $O(n^3)$ can also be achieved using the simple Floyd-Warshall algorithm (see [19]). We next consider the possibility of obtaining faster algorithms for dense graphs.

The all-pairs shortest paths problem is closely related to the $\{\min, +\}$-product of matrices. If $A = (a_{ij})$ and $B = (b_{ij})$ are $n \times n$ matrices, we let $A \star B$ be the $n \times n$ matrix whose (i,j)-th element is $(A \star B)_{ij} = \min_k\{a_{ik} + b_{kj}\}$. We refer to $A \star B$ as the *distance product* of A and B.

Let $G = (V,E)$ be a graph. We may assume that $V = \{1, 2, \ldots, n\}$. Let $W = (w_{ij})$ be an $n \times n$ matrix with $w_{ij} = w(i,j)$, if $(i,j) \in E$, and $w_{ij} = +\infty$, otherwise. It is easy to see that W^n, where the exponentiation is done with respect to distance product, gives the distance between any pair of vertices in the graph. Furthermore, the graph contains a negative cycle if and only if there are negative elements on the diagonal of W^n. Thus, the APSP problem can be easily solved using $O(\log n)$ distance products. In fact, under some reasonable assumptions, this logarithmic factor can be saved, and it can be shown that the APSP problem is not harder than the problem of computing a single distance product of two $n \times n$ matrices (see [1, Theorem 5.7 on p. 204]).

Distance products could be computed naively in $O(n^3)$ time, but this is of no help to us. Algebraic, i.e., $\{+, \times\}$-products of matrices could be computed much faster. Strassen [65] was the first to show that it could be done using $o(n^3)$ operations. Many improvements followed. We let ω be the *exponent* of matrix multiplication, i.e., the smallest constant for which matrix multiplication can be performed using only $O(n^{\omega+o(1)})$ algebraic operations, i.e., additions, *subtractions*, and multiplications. (For brevity, we 'forget' the annoying $o(1)$ term, and use ω as a substitute for $\omega + o(1)$.) Coppersmith and Winograd [18] showed that $\omega < 2.376$. The only known lower bound on ω is the trivial lower bound $\omega \geq 2$.

Could similar techniques be used, directly, to obtain $o(n^3)$ algorithms for computing distance products? Unfortunately not. The fast matrix multiplication algorithms rely in an essential way on the fact that addition operations could be reversed, via subtractions. This opens the way for clever cancellations that speed up the computation. It is known in fact, that matrix multiplication requires $\Omega(n^3)$ operations, if only additions and multiplications are allowed. This follows from lower bounds on monotone circuits for *Boolean matrix multiplication* obtained by Mehlhorn and Galil [53] and by Paterson [56].

Yuval [77] describes a simple transformation from distance products to standard algebraic products. He assumes, however, that exact exponentiations and logarithms of infinite precision real numbers could be computed in constant time. His model, therefore, is very unrealistic. His ideas could be exploited, however, in a more restricted form, as would be mentioned in Section 4.3. (Several erroneous follow-ups of Yuval's result appeared in the 80's. They are not cited here.)

Fredman [29] describes an elegant way of computing distance products of two $n \times n$ matrices using $O(n^{5/2})$ additions and comparisons in the algebraic computation tree model. There does not seem to be any an efficient way of implementing his algorithm in the RAM model, as it require programs of exponential size. However, by running his algorithm on small matrices, for which short enough programs that implement his algorithm could be precomputed, he obtains an $O(n^3(\log\log n/\log n)^{1/3})$ time algorithm for computing distance products. Takaoka [66] slightly improves his bound to $O(n^3(\log\log n/\log n)^{1/2})$.

Is there a genuinely sub-cubic algorithm for the APSP problem in the addition-comparison model, i.e., an algorithm that runs in $O(n^{3-\epsilon})$ time, for some $\epsilon > 0$?

4.3 Integer edge weights – directed graphs

We next consider the all-pairs shortest paths problem in directed graphs with integer edge weights. Even the unweighted case is interesting here. The first to obtain a genuinely sub-cubic algorithm for the unweighted problem were Alon, Galil and Margalit [4]. Their algorithm runs in $\tilde{O}(n^{(3+\omega)/2})$ time.[1] Their result also extends to the case in which the edge weights are in the range $\{0, \ldots, M\}$. The running time of their algorithm is then $\tilde{O}(M^{(\omega-1)/2}n^{(3+\omega)/2})$, if $M \leq n^{(3-\omega)/(\omega+1)}$, and $\tilde{O}(Mn^{(5\omega-3)/(\omega+1)})$, if $M \geq n^{(3-\omega)/(\omega+1)}$ (see Galil and Margalit [36, 37]). Takaoka [67] obtained an algorithm whose running time is $\tilde{O}(M^{1/3}n^{(6+\omega)/3})$. The bound of Takaoka is better than the bound of Alon *et al.* [4] for larger values of M. The running time of Takaoka's algorithm is sub-cubic for $M < n^{3-\omega}$.

[1] We use $\tilde{O}(f)$ as a shorthand for $f \cdot (\log n)^{O(1)}$. In the SSSP problem we are fighting to shave off sub-logarithmic factors. In the APSP problem the real battle is still over the right exponent of n, so we use the $\tilde{O}(\cdot)$ notation to hide not so interesting polylogarithmic factors.

The algorithms of Galil and Margalit [36, 37] and of Takaoka [67] were improved by Zwick [78, 79]. Furthermore, his algorithm works with edge weights to be in the range $\{-M, \ldots, M\}$. The improvement is based on two ingredients. The first is an $\tilde{O}(Mn^\omega)$ algorithm, mentioned in [4], for computing distance products of $n \times n$ matrices whose finite elements are in the range $\{-M, \ldots, M\}$. This algorithm is based on the idea of Yuval [77]. It is implemented this time, however, in a realistic model. The algorithm uses both the fast matrix multiplication algorithm of [18], and the *integer* multiplication algorithm of [62]. (Note that an $\tilde{O}(Mn^\omega)$ time algorithm for distance products does not give immediately an $\tilde{O}(Mn^\omega)$ time algorithm for the APSP problem, as the range of the elements is increased by each distance product.) The second ingredient is a sampling technique that enables the replacement of a distance product of two $n \times n$ matrices by a smaller *rectangular* product. The algorithm uses, therefore, the fast rectangular matrix multiplication algorithm of [17] (see also [44]).

To state the running time of Zwick's algorithm, we need to introduce exponents for rectangular matrix multiplication. Let $\omega(r)$ be the smallest constant such that the product of an $n \times n^r$ matrix by an $n^r \times n$ matrix could be computed using $O(n^{\omega(r)+o(1)})$ algebraic operations. Suppose that $M = n^t$. Then, the running time of his algorithm his $\tilde{O}(n^{2+\mu(t)})$, where $\mu = \mu(t)$ satisfies $\omega(\mu) = 1 + 2\mu - t$. The best available bounds on $\omega(r)$ imply, for example, that $\mu(0) < 0.575$, so that the APSP problem for directed graphs with edge weights taken from $\{-1, 0, 1\}$ can be solved in $O(n^{2.575})$ time. The algorithm runs in sub-cubic time when $M < n^{3-\omega}$, as was the case with Takaoka's algorithm.

The algorithms mentioned above differ from almost all the other algorithms mentioned in this survey in that augmenting them to produce a compact representation of shortest paths, and not only distances, is a non-trivial task. This requires the computation of *witnesses* for Boolean matrix multiplications and distance products. A simple randomized algorithm for computing witnesses for Boolean matrix multiplication is given by Seidel [63]. His algorithm was *derandomized* by Alon and Naor [6] (see also [5]). An alternative, somewhat slower deterministic algorithm was given by Galil and Margalit [35].

Obtaining improved algorithms, and in particular sub-cubic algorithms for larger values of M for this version of the problem is a challenging open problem.

Finally, Hagerup [40] obtained an $O(mn + n \log\log n)$ time algorithm for the problem in the word RAM model. Could this be reduced to $O(mn)$?

4.4 Integer edge weights – undirected graphs

Galil and Margalit [36, 37] and Seidel [63], obtained $\tilde{O}(n^\omega)$ time algorithms for solving the APSP problem for unweighted *undirected* graphs. Seidel's algorithm is much simpler. Both algorithms show, in fact, that this version of the problem is harder then the *Boolean* matrix multiplication problem by at most a logarithmic factor. (Seidel's algorithm, as it appears in [63], uses integer matrix products, but it is not difficult to obtain a version of it that uses only Boolean products.) Again, witnesses for Boolean matrix products are needed, if paths, and not only distances, are to be found.

Seidel's algorithm is extremely simple and elegant. There seems to be no simple way, however, of using his ideas to obtain a similar algorithm for weighted graphs. The algorithm of Galil and Margalit can be extended, in a fairly straightforward way, to handle small integer edge weights. The running time of their algorithm, when the edge weights are taken from $\{0, 1, \ldots, M\}$, is $\tilde{O}(M^{(\omega+1)/2}n^{\omega})$. An improved time bound of $\tilde{O}(Mn^{\omega})$ for the problem was recently obtained by Shoshan and Zwick [64]. They show, in fact, that the APSP problem for undirected graphs with edge weights taken from $\{0, 1, \ldots, M\}$ is harder than the problem of computing the distance product of two $n \times n$ matrices with elements taken from the *same* range by at most a logarithmic factor. (As mentioned in Section 4.3, this is not known for directed graphs.)

Obtaining improved algorithms, and in particular sub-cubic algorithms for larger values of M for this version of the problem is again a challenging open problem. For undirected graphs this is equivalent, as mentioned, to obtaining faster algorithms for distance products of matrices with elements in the range $\{0, 1, \ldots, M\}$.

5 All-pairs shortest paths – approximate results

The cost of exactly computing all distances in a graph may be prohibitively large. In this section we explore the savings that may be obtained by settling for approximate distances and almost shortest paths. Throughout this section, we assume that the edge weights are nonnegative.

We say that an estimated distance $\hat{\delta}(u, v)$ is of *stretch* t if and only if $\delta(u, v) \leq \hat{\delta}(u, v) \leq t \cdot \delta(u, v)$. We say that an estimated distance $\hat{\delta}(u, v)$ is of *surplus* t if and only if $\delta(u, v) \leq \hat{\delta}(u, v) \leq \delta(u, v) + t$. All our estimates correspond to actual paths in the graph, and are thus upper bounds on the actual distances.

5.1 Directed graphs

It is not difficult to see [23] that for any finite t, obtaining stretch t estimates of all distances in a graph is at least as hard as Boolean matrix multiplication. On the other hand, Zwick [78] shows that for any $\epsilon > 0$, approximate distances of stretch $1 + \epsilon$ of all distances in a directed graph may be computed in time $\tilde{O}((n^{\omega}/\epsilon) \log(W/\epsilon))$, where W is the largest edge weight in the graph, after the edge weights are scaled so that the smallest nonzero edge weight is 1.

5.2 Unweighted undirected graphs

Surprisingly, perhaps, when the graph is undirected and unweighted, estimated distance with small *additive* error may be computed rather quickly, *without* using fast matrix multiplication algorithms. This was first shown by Aingworth *et al.* [3]. They showed that surplus 2 estimates of the distances between k *specified* pairs of vertices may be computed in $O(n^{3/2}(k \log n)^{1/2})$ time. In particular, surplus 2 estimates of all the distances in the graph, and corresponding paths, may be computed in $O(n^{5/2}(\log n)^{1/2})$ time. Aingworth *et al.* [3] also give a 2/3-approximation algorithm for the *diameter* of a weighted directed graph that runs in $O(m(n\log n)^{1/2} + n^2 \log n)$ time.

Elkin [25] describes an algorithm for computing estimated distances from a set S of sources to all other vertices of the graph. He shows that for any $\epsilon > 0$, there is a constant $b = b(\epsilon)$, such that an estimated distance $\hat{\delta}(u,v)$ satisfying $\delta(u,v) \le \hat{\delta}(u,v) \le (1+\epsilon)\delta(u,v)+b$, for every $u \in S$ and $v \in V$, may be computed in $O(mn^{\epsilon} + |S|n^{1+\epsilon})$ time. Furthermore, the corresponding shortest paths, use only $O(n^{1+\epsilon})$ edges of the graph. (See also Section 6.2.) Note, however, that although the term multiplying $\delta(u,v)$ above can be made arbitrarily close to 1, the errors in the estimates obtained are not purely additive.

Dor *et al.* [23] obtained improved algorithms for obtaining finite surplus estimates of *all* distances in the graph. They show that surplus 2 estimates may be computed in $\tilde{O}(n^{3/2}m^{1/2})$ time, and also in $\tilde{O}(n^{7/3})$ time. Furthermore, they exhibit a surplus-time tradeoff showing that surplus $2(k-1)$ estimates of all distances may be computed in $\tilde{O}(kn^{2-1/k}m^{1/k})$ time. In particular, surplus $O(\log n)$ estimates of all distances may be obtained in almost optimal $\tilde{O}(n^2)$ time.

5.3 Weighted undirected graphs

Cohen and Zwick [16] adapted the techniques of Dor *et al.* [23] for weighted graphs. They obtain stretch 2 estimates of all distances in $\tilde{O}(n^{3/2}m^{1/2})$ time, stretch 7/3 estimates in $\tilde{O}(n^{7/3})$ time, and stretch 3 estimates, and corresponding stretch 3 paths, in almost optimal $\tilde{O}(n^2)$ time. Algorithms with additive errors are also presented. They show, for example, that if p is a any path between u and v, then the estimate $\hat{\delta}(u,v)$ produced by the $\tilde{O}(n^{3/2}m^{1/2})$ time algorithm satisfies $\hat{\delta}(u,v) \le w(p) + 2w_{\max}(p)$, where $w(p)$ is the length of p, and $w_{\max}(p)$ is the weight of the heaviest edge on p.

6 Spanners

6.1 Weighted undirected graphs

Let $G = (V,E)$ be an undirected graph. In many applications, many of them related to distributed computing (see [57]), it is desired to obtain a sparse subgraph $H = (V,F)$ of G that approximates, at least to a certain extent, the distances in G. Such a subgraph H is said to be a *t-spanner* of G if and only if for every $u,v \in V$ we have $\delta_H(u,v) \le t\cdot\delta_G(u,v)$. (This definition, implicit in [8], appears explicitly in [58].)

Althöfer *et al.* [7] describe the following simple algorithm for constructing a t-spanner of an undirected graph $G = (V,E)$ with nonnegative edge weights. The algorithm is similar to Kruskal's algorithm [51] (see also [19]) for computing minimum spanning trees: Let $F \leftarrow \phi$. Consider the edges of G in nondecreasing order of weight. If $(u,v) \in E$ is the currently considered edge and $w(u,v) < t\cdot\delta_F(u,v)$, then add (u,v) to F. It is easy to see that at the end of this process

$H = (V, F)$ is indeed a t-spanner of G. It is also easy to see that the *girth* of H is greater than $t + 1$. (The girth of a graph G is the smallest number of edges on a cycle in G.) It is known that any graph with at least $n^{1+1/k}$ edges contains a cycle with at most $2k$ edges. It follows that any weighted graph on n vertices has a $(2k-1)$-spanner with $O(n^{1+1/k})$ edges. This result is believed to be tight for any $k \geq 1$. It is proved, however, only for $k = 1, 2, 3$ and 5. (See, e.g., [73].)

The fastest known implementation of the algorithm of Althöfer *et al.* [7] runs in $O(mn^{1+1/k})$ time. If the graph is unweighted, then a $(2k-1)$-spanner of size $O(n^{1+1/k})$ can be easily found in $O(m)$ time [41]. Thorup and Zwick [73], improving a result of Cohen [14], give a randomized algorithm for computing a $(2k-1)$-spanner of size $O(n^{1+1/k})$ in $O(kmn^{1/k})$ expected time.

Approximation algorithms and hardness results related to spanners were obtained by Kortsarz and Peleg [50] and Elkin and Peleg [27].

6.2 Unweighted undirected graphs

Following Elkin and Peleg [26], we say that a subgraph H of an unweighted graph G is an (a, b)-spanner of G if and only if $\delta_H(u, v) \leq a \cdot \delta(u, v) + b$, for every $u, v \in V$. (For a related notion of k-emulators, see [23].) Elkin and Peleg [26] and Elkin [25], improving and extending some preliminary results of Dor *et al.* [23], show that any graph on n vertices has a $(1, 2)$-spanner with $O(n^{3/2})$ edges, and that for any $\epsilon > 0$ and $\delta > 0$ there exists $b = b(\epsilon, \delta)$, such that every graph on n vertices has a $(1 + \epsilon, b)$-spanner with $O(n^{1+\delta})$ edges.

The intriguing open problem here is whether the $(1+\epsilon, b)$-spanners of [26, 25] could be turned into $(1, b)$-spanners, i.e., purely additive spanners. In particular, it is still open whether there exists a $b > 0$ such that any graph on n vertices has a $(1, b)$-spanner with $o(n^{3/2})$ edges. (In [23] it is shown that any graph on n vertices has a *Steiner* $(1, 4)$*-spanner* with $\tilde{O}(n^{4/3})$ edges. A Steiner spanner, unlike standard spanners, is not necessarily a subgraph of the approximated graph. Furthermore, the edges of the Steiner spanner may be weighted, even if the original graph is unweighted.)

7 Distance oracles

In this section we consider the following problem: We are given a graph $G = (V, E)$. We would like to *preprocess* it so that subsequent *distance queries* or *shortest path queries* could be answered very quickly. A naive solution is to solve the APSP problem, using the best available algorithm, and store the $n \times n$ matrix of distances. Each distance query can then be answered in constant time. The obvious drawbacks of this solution are the large preprocessing time and large space requirements. Much better solutions exist when the graph is undirected, and when we are willing to settle for approximate results.

The term *approximate distance oracles* is coined in Thorup and Zwick [73], though the problem was considered previously by Awerbuch *et al.* [9], Cohen [14] and Dor *et al.* [23]. Improving the results of these authors, Thorup and Zwick [73] show that for any $k \geq 1$, a graph $G = (V, E)$ on n vertices can be preprocessed in $O(kmn^{1/k})$ expected time, constructing a data structure of size $O(kn^{1+1/k})$, such that a stretch $2k-1$ answer to any distance query can be produced in $O(k)$ time. The space requirements of these approximate distance oracles are optimal for $k = 1, 2, 3, 5$, and are conjectured to be optimal for any value of k. (This is related to the conjecture regarding the size of $(2k-1)$-spanners made in Section 6.1. See discussion in [73].)

Many open problems still remain regarding the possible tradeoffs between the preprocessing time, space requirement, query answering time, and the obtained stretch of approximate distance oracles. In particular, is it possible to combine the techniques of [16] and [73] to obtain a stretch 3 distance oracle with $\tilde{O}(n^2)$ preprocessing time, $\tilde{O}(n^{3/2})$ space, and constant query time? Which tradeoffs are possible when no randomization is allowed? Finally, all the distance oracles currently available have multiplicative errors. Are there non-trivial distance oracles with additive errors?

Addendum

After the preparation of the camera ready version of the survey, a paper by Pettie and Ramachandran [59] was brought to my attention. Pettie and Ramachandran [59] also define the addition-comparison model and make a further distinction between RAM and pointer machine algorithms. Their main result is an algorithm for the all-pairs shortest paths (APSP) problem for *undirected* graphs which runs on a pointer machine in $O(mn\alpha(m,n))$ time while making only $O(mn \log \alpha(m,n))$ additions and comparisons. (Here m is the number of edges, n is the number of vertices, and $\alpha(m,n)$ is Tarjan's inverse-Ackermann function.) The main ingredient used to obtain this new algorithm is a single-source shortest paths (SSSP) algorithm for undirected graphs with positive edge weights that runs in $O(m\alpha(m,n) + n \log\log R)$ time on a pointer machine, where R is the ratio between the maximum and minimum edge weights. They also obtain an algorithm for the SSSP problem in directed graphs that runs in $O(m + n \log R)$ time on a pointer machine.

References

1. A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, 1974.
2. R.K. Ahuja, K. Mehlhorn, J.B. Orlin, and R.E. Tarjan. Faster algorithms for the shortest path problem. *Journal of the ACM*, 37:213–223, 1990.
3. D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28:1167–1181, 1999.
4. N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54:255–262, 1997.

5. N. Alon, Z. Galil, O. Margalit, and M. Naor. Witnesses for boolean matrix multiplication and for shortest paths. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania*, pages 417–426, 1992.
6. N. Alon and M. Naor. Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions. *Algorithmica*, 16:434–449, 1996.
7. I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993.
8. B. Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32:804–823, 1985.
9. B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Near-linear time construction of sparse neighborhood covers. *SIAM Journal on Computing*, 28:263–277, 1999.
10. M. Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, Boston, Massachusetts*, pages 80–86, 1983.
11. B.V. Cherkassky, A.V. Goldberg, and T. Radzik. Shortest paths algorithms: theory and experimental evaluation. *Mathematical Programming (Series A)*, 73(2):129–174, 1996.
12. B.V. Cherkassky, A.V. Goldberg, and C. Silverstein. Buckets, heaps, lists, and monotone priority queues. *SIAM Journal on Computing*, 28(4):1326–1346, 1999.
13. E. Cohen. Using selective path-doubling for parallel shortest-path computations. *Journal of Algorithms*, 22:30–56, 1997.
14. E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal on Computing*, 28:210–236, 1999.
15. E. Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *Journal of the ACM*, 47(1):132–166, 2000.
16. E. Cohen and U. Zwick. All-pairs small-stretch paths. *Journal of Algorithms*, 38:335–353, 2001.
17. D. Coppersmith. Rectangular matrix multiplication revisited. *Journal of Complexity*, 13:42–49, 1997.
18. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.
19. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to algorithms*. The MIT Press, 1990.
20. L.J. Cowen. Compact routing with minimum stretch. *Journal of Algorithms*, 38:170–183, 2001.
21. C. Demetrescu and G. F. Italiano. Fully dynamic transitive closure: breaking through the $O(n^2)$ barrier. In *Proceedings of the 41th Annual IEEE Symposium on Foundations of Computer Science, Redondo Beach, California*, pages 381–389, 2000.
22. E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
23. D. Dor, S. Halperin, and U. Zwick. All pairs almost shortest paths. *SIAM Journal on Computing*, 29:1740–1759, 2000.
24. J.R. Driscoll, H.N. Gabow, R. Shrairman, and R.E. Tarjan. Relaxed heaps: an alternative to Fibonacci heaps with applications to parallel computation. *Communications of the ACM*, 31(11):1343–1354, 1988.
25. M.L. Elkin. Computing almost shortest paths. Technical Report MCS01-03, Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel, 2001.

26. M.L. Elkin and D. Peleg. $(1+\epsilon, \beta)$-Spanner constructions for general graphs. In *Proceedings of the 33th Annual ACM Symposium on Theory of Computing, Crete, Greece*, 2001. To appear.
27. M.L. Elkin and D. Peleg. Approximating k-spanner problems for $k > 2$. In *Proceedings of the 8th Conference on Integer Programming and Combinatorial Optimization, Utrecht, The Netherlands*, 2001. To appear.
28. D. Eppstein. Spanning trees and spanners. In Jörg-Rudiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, chapter 9, pages 425–461. Elsevier, 2000.
29. M.L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing*, 5:49–60, 1976.
30. M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.
31. M.L. Fredman and D.E. Willard. Surpassing the information-theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993.
32. M.L. Fredman and D.E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences*, 48:533–551, 1994.
33. H.N. Gabow. Scaling algorithms for network problems. *Journal of Computer and System Sciences*, 31(2):148–168, 1985.
34. H.N. Gabow and R.E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18:1013–1036, 1989.
35. Z. Galil and O. Margalit. Witnesses for boolean matrix multiplication. *Journal of Complexity*, 9:201–221, 1993.
36. Z. Galil and O. Margalit. All pairs shortest distances for graphs with small integer length edges. *Information and Computation*, 134:103–139, 1997.
37. Z. Galil and O. Margalit. All pairs shortest paths for graphs with small integer length edges. *Journal of Computer and System Sciences*, 54:243–254, 1997.
38. A.V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24:494–504, 1995.
39. T. Hagerup. Sorting and searching on the word RAM. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science, Paris, France*, pages 366–398, 1998.
40. T. Hagerup. Improved shortest paths on the word RAM. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming, Geneva, Switzerland*, pages 61–72, 2000.
41. S. Halperin and U. Zwick. Unpublished result, 1996.
42. Y. Han. Improved fast integer sorting in linear space. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, Washington, D.C.*, pages 793–796, 2001.
43. M.R. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55:3–23, 1997.
44. X. Huang and V.Y. Pan. Fast rectangular matrix multiplications and applications. *Journal of Complexity*, 14:257–299, 1998.
45. D.B. Johnson. Efficient algorithms for shortest paths in sparse graphs. *Journal of the ACM*, 24:1–13, 1977.
46. D.R. Karger, D. Koller, and S.J. Phillips. Finding the hidden path: time bounds for all-pairs shortest paths. *SIAM Journal on Computing*, 22:1199–1217, 1993.
47. R.M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

48. V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, New York, New York*, pages 81–89, 1999.
49. P.N. Klein and S. Subramanian. A randomized parallel algorithm for single-source shortest paths. *Journal of Algorithms*, 25(2):205–220, 1997.
50. G. Kortsarz and D. Peleg. Generating sparse 2-spanners. *Journal of Algorithms*, 17(2):222–236, 1994.
51. J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.
52. C.C. McGeoch. All-pairs shortest paths and the essential subgraph. *Algorithmica*, 13:426–461, 1995.
53. K. Mehlhorn and Z. Galil. Monotone switching circuits and Boolean matrix product. *Computing*, 16(1-2):99–111, 1976.
54. J.S.B. Mitchell. Shortest paths and networks. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 445–466. CRC Press LLC, Boca Raton, FL, 1997.
55. J.S.B. Mitchell. Geometric shortest paths and network optimization. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
56. M.S. Paterson. Complexity of monotone networks for Boolean matrix product. *Theoretical Computer Science*, 1(1):13–20, 1975.
57. D. Peleg. *Distributed computing – A locality-sensitive approach.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.
58. D. Peleg and A.A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13:99–116, 1989.
59. S. Pettie and V. Ramachandran. Computing shortest paths with comparisons and additions. Technical Report UTCS TR-01-12, Department of Computer Sciences, The University of Texas at Austin, 2001. http://www.cs.utexas.edu/users/vlr/papers/sp01.ps.
60. R. Raman. Priority queues: small, monotone and trans-dichotomous. In *Proceedings of the 4nd European Symposium on Algorithms, Barcelona, Spain*, pages 121–137, 1996.
61. R. Raman. Recent results on the single-source shortest paths problem. *SIGACT News*, 28:81–87, 1997.
62. A. Schönhage and V. Strassen. Schnelle multiplikation grosser zahlen. *Computing*, 7:281–292, 1971.
63. R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51:400–403, 1995.
64. A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, New York, New York*, pages 605–614, 1999.
65. V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.
66. T. Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *Information Processing Letters*, 43:195–199, 1992.
67. T. Takaoka. Subcubic cost algorithms for the all pairs shortest path problem. *Algorithmica*, 20:309–318, 1998.
68. M. Thorup. Faster deterministic sorting and priority queues in linear space. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California*, pages 550–555, 1998.

69. M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46:362–394, 1999.
70. M. Thorup. Floats, integers, and single source shortest paths. *Journal of Algorithms*, 35:189–201, 2000.
71. M. Thorup. On RAM priority queues. *SIAM Journal on Computing*, 30(1):86–109, 2000.
72. M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. manuscript, 2001.
73. M. Thorup and U. Zwick. Approximate distance oracles. In *Proceedings of the 33th Annual ACM Symposium on Theory of Computing, Crete, Greece*, pages 183–192, 2001.
74. M. Thorup and U. Zwick. Compact routing schemes. In *Porceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures, Crete, Greece*, pages 1–10, 2001.
75. P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6(3):80–82, 1977.
76. P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977.
77. G. Yuval. An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications. *Information Processing Letters*, 4:155–156, 1976.
78. U. Zwick. All pairs shortest paths in weighted directed graphs – exact and almost exact algorithms. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, California*, pages 310–319, 1998. Journal version submitted for publicaiton under the title *All-pairs shortest paths using bridging sets and rectangular matrix multiplication.*
79. U. Zwick. All pairs lightest shortest paths. In *Proceedings of the 31th Annual ACM Symposium on Theory of Computing, Atlanta, Georgia*, pages 61–69, 1999.