All-Pairs Shortest Paths with Real Weights in $O(n^3/\log n)$ Time

Timothy M. Chan*

May 10, 2005

Abstract

We describe an $O(n^3/\log n)$-time algorithm for the all-pairs-shortest-paths problem for a real-weighted directed graph with n vertices. This slightly improves a series of previous, slightly subcubic algorithms by Fredman (1976), Takaoka (1992), Dobosiewicz (1990), Han (2004), Takaoka (2004), and Zwick (2004). The new algorithm is surprisingly simple and different from previous ones.

1 Introduction

The *all-pairs-shortest-paths* problem (APSP) is of course one of the most well-studied problems in algorithm design. We consider here the general case where the input is a weighted directed graph and edge weights are arbitrary real numbers. The problem is to compute the shortest-path distance between every pair of vertices, together with a representation of these shortest paths (so that the shortest path for any given vertex pair can be retrieved in time linear in its length).

The classical Floyd–Warshall algorithm [6] solves the APSP problem in $O(n^3)$ time for a graph with n vertices. Fredman [10] was the first to realize that subcubic running time is attainable: he gave an algorithm with an impressive-looking time bound of $O(n^3(\log\log n/\log n)^{1/3})$. Later, Takaoka [19] and Dobosiewicz [7] refined Fredman's approach and reduced the bound to $O(n^3\sqrt{\log\log n/\log n})$ and $O(n^3/\sqrt{\log n})$ respectively. Just last year, several interesting, independent developments have occurred: first Han [12] announced an improved $O(n^3(\log\log n/\log n)^{5/7})$-time algorithm, then Takaoka [20] announced an even better $O(n^3(\log\log n)^2/\log n)$-time algorithm, and finally Zwick [23] found the algorithm with the current record of $O(n^3\sqrt{\log\log n}/\log n)$ time. The record turns out to be short-lived—in this note, we obtain yet another algorithm with a further improved running time of $O(n^3/\log n)$.

Related work. For *sparse* graphs, a more efficient solution to APSP is to apply Dijkstra's single-source algorithm n times, as described in any decent algorithm textbook [6]. Using a Fibonacci-heap implementation (with Johnson's preprocessing step if negative weights are allowed), the running time is $O(n^2\log n + mn)$, where m denotes the number of edges. For a long time, this was the best result known, until recently Pettie and Ramachandran [14] and Pettie [13] have managed to bring the time bounds down to $O(mn\log\alpha(m,n))$ and $O(n^2\log\log n + mn)$ for undirected and directed graphs, respectively, using rather complicated techniques.

*School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada (tmchan@uwaterloo.ca).

A flurry of activities in the last decade has concentrated on the case of graphs with *small integer weights* (and, in particular, unweighted graphs), where a number of genuinely subcubic algorithms [2, 11, 16, 17, 22] have been developed using known methods for matrix multiplication over rings [5, 18]. Currently, the best such APSP algorithms for undirected and directed graphs run in $O(n^{2.376}M)$ and $O(n^{2.575}M^{0.681})$ time respectively [17, 22], where M denotes the maximum edge weight (in absolute value). Note that these running times are subcubic only when $M \ll n^{0.634}$. It is not known whether such matrix multiplication methods can help for APSP in the case of real weights, or for that matter, integer weights from the range $\{0, 1, \ldots, n\}$. (Even if the answer is affirmative, algorithms that involve so-called "fast" matrix multiplication are not necessarily attractive from a practical point of view.) Feder and Motwani [9] described an $O(n^3/\log n)$-time algorithm that avoids fast matrix multiplication but their algorithm works only for unweighted, undirected graphs.

About the new algorithm. We confess that our result represents only a minute improvement over previous slightly subcubic algorithms in the general real-weight case—a mere $\sqrt{\log\log n}$-factor speedup over the previous result by Zwick! However, we believe that our algorithm is interesting, because (i) it is conceptually very simple (note the length of the paper) and (ii) it is markedly different from previous approaches:

The approach originated by Fredman [10] (and later continued by Takaoka [19]) broke the $O(n^3)$ barrier by relying on *table-lookup* tricks (storing solutions to all small-sized subproblems in an array for later retrieval in constant time). Dobosiewicz's approach [7] avoided explicit table lookups by exploiting *word-RAM operations* (specifically, performing bitwise-logical operations on $(\log n)$-bit words in unit time). The recent algorithms by Han [12], Takaoka [20], and Zwick [23] all involved even more complicated combinations of approaches. For example, Zwick [23] nontrivially combined Dobosiewicz's approach with a known table-lookup technique for Boolean matrix multiplication [3], resulting in an algorithm that uses both table lookups and word-RAM operations. In contrast, our approach uses *neither* table lookups nor word operations! In fact, our algorithm is readily implementable within the pointer-machine model.[1] Curiously, our approach is *geometrically* inspired (based on a multidimensional divide-and-conquer technique commonly seen in computational geometry). Considering the long history of the APSP problem, it is amusing that this little idea alone can beat all previous algorithms for arbitrary, real-weighted, dense graphs.

2 A Geometric Subproblem

We begin with what may at first appear to be a complete digression: a problem in computational geometry, concerning a special case of off-line orthogonal range searching (also similar to the "maxima" problem) [15]. The problem is to find all *dominating pairs* between a red point set and a blue point set in d-dimensional space, where a red point $p = (p_1, \ldots, p_d) \in \mathbb{R}^d$ and a blue point $q = (q_1, \ldots, q_d) \in \mathbb{R}^d$ are said to form a dominating pair iff $p_k \le q_k$ for all $k = 1, \ldots, d$. The algorithm in the lemma below is standard [15], but unlike traditional analysis in computational geometry, we are interested in the case where the dimension is not a constant.

[1] To be fair, we should mention that some algorithms based on table lookups can be modified to run on pointer machines, for example, by using variants of radix sort [4]. The same could be true for some of the previous APSP algorithms, but such a modification would seem to require much additional effort.

Lemma 2.1 *Given n red/blue points in $\mathbb{R}^d$, we can report all K dominating pairs in $O(c_\varepsilon^d n^{1+\varepsilon}+K)$ time for any constant $\varepsilon \in (0,1)$, where $c_\varepsilon := 2^\varepsilon/(2^\varepsilon - 1)$.*

Proof: We describe a simple divide-and-conquer algorithm. If $n = 1$, we stop. If $d = 0$, we just output all pairs of red and blue points. Otherwise, we compute the median z of the d-th coordinates of all points and let $P_{\mathrm{left},\gamma}$ (resp. $P_{\mathrm{right},\gamma}$) denote the subset of all points of color γ with d-th coordinates at most z (resp. at least z). (Note that we can avoid a linear-time median-finding algorithm if we pre-sort all the d-th coordinates.) We then recursively solve the problem for $P_{\mathrm{left,red}} \cup P_{\mathrm{left,blue}}$, for $P_{\mathrm{right,red}} \cup P_{\mathrm{right,blue}}$, and finally for the projection of $P_{\mathrm{left,red}} \cup P_{\mathrm{right,blue}}$ to the first $d-1$ coordinates. (Note that we can avoid actually projecting the points, by just ignoring the d-th coordinates.) Correctness is immediate.

Excluding the output cost, the running time obeys the recurrence

$$T_d(n) \le 2T_d(n/2) + T_{d-1}(n) + O(n),$$

with $T_d(1) = O(1)$ and $T_0(n) = O(n)$. The additional output cost is bounded by $O(K)$, since each pair is reported once.

Naively, one can establish by induction on d that $T_d(n) = O(n \log^d n)$, yielding an $O(n \log^d n + K)$-time algorithm. This result is already known. (In fact, it is known that one can save one or two logarithmic factors by handling the base cases $d = 1$ and $d = 2$ directly.)

We offer an alternative analysis of the recurrence that is better for certain non-constant values of d and is thus slightly more effective for the application in the next section. We make a change of variable: fixing a parameter b and letting $T'(N) := \max_{b^d n \le N} T_d(n)$, we have

$$T'(N) \le 2T'(N/2) + T'(N/b) + cN,$$

for some constant c. This single-variable recurrence can be solved by standard techniques. For example, by induction, the bound $T'(N) \le c'[N^{1+\varepsilon} - N]$ follows from $T'(N) \le 2c'[(N/2)^{1+\varepsilon} - N/2] + c'[(N/b)^{1+\varepsilon} - N/b] + cN$, as long as the constant c' is sufficiently large, and

$$2/2^{1+\varepsilon} + 1/b^{1+\varepsilon} = 1,$$

which holds by setting $b := c_\varepsilon^{1/(1+\varepsilon)}$. Thus, $T'(N) = O(N^{1+\varepsilon})$, implying $T_d(n) = O((b^d n)^{1+\varepsilon}) = O(c_\varepsilon^d n^{1+\varepsilon})$, and the lemma follows. □

(We note in passing that the above two-variable recurrence can also be recast to fit the type studied by Eppstein [8], by considering the exponential function $T''(r,d) := T_d(2^r)$.)

3 The APSP Algorithm

We are now ready to present our new APSP algorithm. Like previous algorithms, we employ a well-known reduction from APSP to the computation of the *distance product* (also known as the *min-plus product*) of two $n \times n$ matrices: given matrices $A = \langle a_{ik} \rangle_{i,k=1,\ldots,n}$ and $B = \langle b_{kj} \rangle_{k,j=1,\ldots,n}$, the result of this multiplication is defined as the matrix $C = \langle c_{ij} \rangle_{i,j=1,\ldots,n}$ with $c_{ij} := \min_k (a_{ik} + b_{kj})$. Given an algorithm for the distance product, we can solve the APSP problem by repeated squaring [6], but this would increase the running time by a logarithmic factor (which we obviously cannot afford); instead, we apply the reduction described in the text by Aho *et al.* [1, Section 5.9, Corollary 2], which avoids the extra logarithmic factor. It is thus sufficient to upper-bound the complexity of the distance product problem. We emphasize that Strassen's matrix multiplication method and its relatives cannot be applied directly, because in the min-plus case, elements only form a semi-ring.

For notational simplicity, we assume that the minimum term in the expression $\min_k(a_{ik} + b_{kj})$ is unique. (General perturbation techniques can ensure this but are not necessary if we break ties in a consistent manner.) We note that our algorithm can automatically identify the index k attaining the minimum for each c_{ij}. (This property is required so that one can not only determine each shortest path distance but retrieve each shortest path.)

In the following lemma, we reveal the key connection between our earlier geometric problem and distance products of rectangular matrices:

Lemma 3.1 *We can compute the distance product of an $n \times d$ matrix A and a $d \times n$ matrix B in $O(dc_\varepsilon^d n^{1+\varepsilon} + n^2)$ time.*

Proof: The outline of the algorithm is simple: For each $k = 1, \ldots, d$, we compute the set of pairs $X_k = \{(i,j) \mid \forall k' = 1, \ldots, d,\ a_{ik} + b_{kj} \leq a_{ik'} + b_{k'j}\}$; we then set $c_{ij} = a_{ik} + b_{kj}$ for every $(i,j) \in X_k$.

We first make an obvious observation (used also in previous approaches): $a_{ik} + b_{kj} \leq a_{ik'} + b_{k'j}$ is equivalent to $a_{ik} - a_{ik'} \leq b_{k'j} - b_{kj}$. We now make the next observation (which was missed in previous approaches): computing X_k for a fixed k corresponds exactly to computing all dominating pairs between the two d-dimensional point sets

$$\{(a_{ik} - a_{i1}, a_{ik} - a_{i2}, \ldots, a_{ik} - a_{id})\}_{i=1,\ldots,n} \quad \text{and} \quad \{(b_{1j} - b_{kj}, b_{2j} - b_{kj}, \ldots, b_{dj} - b_{kj})\}_{j=1,\ldots,n}.$$

(The dimension is actually $d - 1$, since the k-th coordinates are all 0's.) By Lemma 2.1, this computation takes $O(c_\varepsilon^d n^{1+\varepsilon} + |X_k|)$ time for each k. Since $\sum_{k=1}^d |X_k| = n^2$, the lemma follows. □

The highlight of our approach is already over. To get a subcubic algorithm for the distance product of square matrices, and consequently for APSP, all that remains is to choose an appropriate value for the dimensional parameter d:

Theorem 3.2 *We can compute the distance product of two $n \times n$ matrices in $O(n^3/\log n)$ time.*

Proof: We split the first matrix into n/d matrices $A_1, \ldots, A_{n/d}$ of dimension $n \times d$, and the second matrix into n/d matrices $B_1, \ldots, B_{n/d}$ of dimension $d \times n$. We compute the distance product of A_ℓ and B_ℓ for each $\ell = 1, \ldots, n/d$, by Lemma 3.1, and return the element-wise minimum of these n/d matrices of dimension $n \times n$. The total time is at most the time bound of Lemma 3.1 multiplied by n/d, i.e.,

$$O\left(c_\varepsilon^d n^{2+\varepsilon} + \frac{n^3}{d}\right).$$

The theorem follows by choosing d to be $\log n$ times a sufficiently small constant (depending on $\varepsilon \in (0,1)$). For example, we can set $\varepsilon \approx 0.38$ (with $c_\varepsilon \approx 4.32$) and $d \approx 0.42 \ln n$, to minimize the constant factor in the dominant term. □

Corollary 3.3 *We can solve the APSP problem in $O(n^3/\log n)$ time.*

We conclude by mentioning how easy it is to adapt our algorithm to run on pointer machines: Lemma 2.1 poses no problem by using linked lists. In Lemma 3.1, for each pair $(i, j) \in X_k$, we cannot directly set the value of c_{ij} since random access is forbidden; instead, we insert the pair (j, k) into i's "bucket". Afterwards, for each i, we sort its bucket according to the j value (by scanning through all pairs (j, k) in the bucket, putting the index k into j's "slot", and collecting all slots at the end). We can then set c_{ij} for every i and j, all within $O(n^2)$ time.

4 Discussion

We have demonstrated that a slightly subcubic time bound for the general APSP problem with real weights can be obtained without "cheating" on the RAM via table lookups or word operations, and without algebraic techniques for fast matrix multiplication. Although we have taken a geometric approach, the resulting algorithm shares some similarities with previous algorithms: for example, like in our proof of Lemma 3.1, Fredman's algorithm and its successors use the same primitive operation on the weights (comparing values each of which is the difference of two entries from a common row or column); in addition, Dobosiewicz's algorithm also goes through each index k and compute the same set X_k of index pairs (but in a different way, of course).

Among the series of slightly sub-cubic upper bounds obtained, $O(n^3/\log n)$ looks the most "natural", and it is interesting to contemplate whether we have reached the limit, at least as far as nonalgebraic algorithms are concerned. In any case, reducing the running time further by more than a logarithmic factor would be difficult: even for the simpler problem of *Boolean matrix multiplication*, the best known algorithm without algebraic techniques [3] runs in $O(n^3/\log^2 n)$ time and has not be improved for over three decades.

Although our algorithm is simple enough for implementation, it is primarily of theoretical interest. Some preliminary experiments seem to indicate that even for n about 1000 (where the size of the graph is on the order of a million), the best choice of d is still 2 (i.e., the dimension for the geometric subproblems is 1). Compared to the naive cubic method for computing distance products, the $\log n$-factor speedup can only be "felt" when the input size is very large, but in such cases, caching and other issues become more important.

An interesting theoretical question is whether a similar log-factor-type speedup is possible for sparse graphs. For example, for the simpler problem of computing the *transitive closure* of an unweighted directed graph, Yuster and Zwick in a recent paper [21] asked for an $o(mn)$-time algorithm, but an $O(mn/\log n + n^2)$ time bound is actually easy to get on the word RAM.[2] Can a similar $o(mn)$ running time be obtained for APSP for real-weighted graphs? (The author has recently made progress on this question for the case of unweighted, undirected graphs.)

Finally, we remark that in his original paper [10], Fredman was concerned with decision-tree complexities and found a (nonalgorithmic) way to solve the general APSP problem using $O(n^{2.5})$ comparisons of sums of edge weights (which then led to his slightly subcubic algorithmic result). It remains an open problem to find improved upper bounds or nontrivial lower bounds on the number of comparisons required.

[2]Proof: Assume that the graph is acyclic, since we can precompute the strongly connected components in linear time and contract each component. We want to find the set S_u of all vertices reachable from each vertex u. For each vertex u in reverse topological order, we can compute S_u by taking the union of S_v over all vertices v incident from u. Each of these $O(m)$ set-union operations can be carried out in $O(n/\log n)$ time by representing a set as an $(n/\log n)$-word vector and by using the bitwise-or operation.

Acknowledgement. I thank a reviewer for bringing up references [12, 20] to my attention. This work has been supported by NSERC.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1974.

[2] N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Sys. Sci.*, 54:255–262, 1997.

[3] V. L. Arlazarov, E. C. Dinic, M. A. Kronrod, and I. A. Faradzev. On economical construction of the transitive closure of a directed graph. *Soviet Math. Dokl.*, 11:1209–1210, 1970.

[4] A. L. Buchsbaum, H. Kaplan, A. Rogers, and J. R. Westbrook. Linear-time pointer-machine algorithms for least common ancestors, MST verification, and dominators. In *Proc. 30th ACM Sympos. Theory Comput.*, pages 279–288, 1998.

[5] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Comput.*, 9:251–280, 1990.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* McGraw-Hill, 2nd ed., 2001.

[7] W. Dobosiewicz. A more efficient algorithm for the min-plus multiplication. *Int. J. Computer Math.*, 32:49–60, 1990.

[8] D. Eppstein. Quasiconvex analysis of backtracking algorithms. In *Proc. 15th ACM-SIAM Sympos. Discrete Algorithms*, pages 788–797, 2004.

[9] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. *J. Comput. Sys. Sci.*, 51:261–272, 1995.

[10] M. L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5:49–60, 1976.

[11] Z. Galil and O. Margalit. All pairs shortest paths for graphs with small integer length edges. *J. Comput. Sys. Sci.*, 54:243–254, 1997.

[12] Y. Han. Improved algorithm for all pairs shortest paths. *Inform. Process. Lett.*, 91:245–250, 2004.

[13] S. Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theoret. Comput. Sci.*, 312:47–74, 2004.

[14] S. Pettie and V. Ramachandran. A shortest path algorithm for real-weighted undirected graphs. *SIAM J. Comput.*, to appear.

[15] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[16] R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Sys. Sci.*, 51:400–403, 1995.

[17] A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proc. 40th IEEE Sympos. Found. Comput. Sci.*, pages 605–614, 1999.

[18] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[19] T. Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *Inform. Process. Lett.*, 43:195–199, 1992.

[20] T. Takaoka. A faster algorithm for the all-pairs shortest path problem and its application. In *Proc. 10th Int. Conf. Comput. Comb.*, Lect. Notes Comput. Sci., vol. 3106, Springer-Verlag, pages 278–289, 2004.

[21] R. Yuster and U. Zwick. Fast sparse matrix multiplication. In *Proc. 12th European Sympos. Algorithms*, Lect. Notes Comput. Sci., vol. 3221, Springer-Verlag, pages 604–615, 2004.

[22] U. Zwick. All-pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49:289–317, 2002.

[23] U. Zwick. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. In *Proc. 15th Int. Sympos. Algorithms and Computation*, Lect. Notes Comput. Sci., vol. 3341, Springer-Verlag, pages 921–932, 2004.