All-Pairs Shortest-Paths Computation in the Presence of Negative Cycles

Kurt Mehlhorn Volker Priebe Guido Schäfer
Naveen Sivadasan

Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany

Abstract

We present an algorithm that solves the all-pairs shortest-paths problem on a directed graph with n vertices and m arcs in time $O(nm + n^2 \log n)$, where the arcs are assigned real, possibly negative costs. Our algorithm is new in the following respect. It computes the distance $\mu(v, w)$ between each pair v, w of vertices even in the presence of negative cycles, where $\mu(v, w)$ is defined as the infimum of the costs of all directed paths from v to w.

Key words: graph algorithms, shortest-paths algorithms

1 Introduction

Given a directed graph in which arcs are assigned real costs, let the cost of a path be the sum of the costs of its arcs. *Shortest-paths problems* ask for *distances* between pairs of vertices, where we define, for any two vertices v, w, the distance $\mu(v, w)$ of vertex w from vertex v as the infimum of the costs of all directed paths from v to w. All distances are finite real numbers if and only if the graph is strongly connected and does not contain any *negative cycles*, that is, directed cycles of negative cost. If w cannot be reached by any directed path from v, then $\mu(v, w) = +\infty$; if there is a path from v to w that contains a negative cycle, then $\mu(v, w) = -\infty$, since paths from v to w can traverse this cycle arbitrarily often. We consider two types of shortest-paths

Email addresses: mehlhorn@mpi-sb.mpg.de (Kurt Mehlhorn), priebe@mpi-sb.mpg.de (Volker Priebe), schaefer@mpi-sb.mpg.de (Guido Schäfer), ns@mpi-sb.mpg.de (Naveen Sivadasan).

problems (with respect to the generalized distances we just defined). In the *single-source shortest-paths problem*, we are interested in the distances of all vertices from a given source vertex; in the *all-pairs shortest-paths problem*, we want to compute the distances between all pairs of vertices.

For a given directed graph $G = (V, A)$, let n and m denote the cardinality of the vertex set V and of the arc set A, respectively. We write (G, c) for G with arc costs c. We present an algorithm that solves the all-pairs shortest-paths problem on (G, c) in time $O(nm + n^2 \log n)$.

2 Complexity of known algorithms

The worst-case complexity of known algorithms for the *single-source shortest-paths problem* depends heavily on whether or not arc costs are allowed to be negative. In fact, if all arc costs are non-negative, then Dijkstra's algorithm [3] solves the single-source shortest-paths problem in time $O(m + n \log n)$, if the algorithm is implemented with efficient data structures as described, for example, in [6,2]. In the general case of possibly negative arc costs, a slightly enhanced version of the Bellman–Ford algorithm [1,5] solves the single-source shortest-paths problem in time $O(nm)$ even if one allows distances of $\pm\infty$; see [8, Section 7.5.7]. Clearly, a run of this version of the Bellman–Ford algorithm also detects a negative cycle in G, if one exists.

The *all-pairs shortest-paths problem* on graphs with real arc costs can be solved by n calls of the Bellman–Ford algorithm, one for each source vertex, which needs time $O(n^2 m)$ in total. As for the all-pairs shortest-paths problem on graphs (G, c) that do *not* contain any negative cycles, the solution of *one* single-source shortest-paths problem allows us to transform a problem with arbitrary real arc costs c into an equivalent problem with non-negative arc costs [11,4,9]. Namely, construct an auxiliary graph G' by adding a new vertex s' to V that is connected to all vertices v of G by an arc (s', v) of zero cost. Surely, G' does not contain any negative cycles, since G does not. The distances $\pi'(v)$ in G' of v from s', for any vertex $v \in V$, can be computed by one call of the Bellman–Ford algorithm. By the optimality conditions for the single-source shortest-paths problem, we have

$$\pi'(w) \leq \pi'(v) + c(v, w) \ , \quad \text{for all } (v, w) \in A \ , \tag{1}$$

where $c(v, w)$ denotes the cost of arc (v, w). (In fact, values $\pi'(v)$, for all $v \in V$, that satisfy (1) exist if and only if (G, c) does not contain any negative cycles.) It follows directly from (1) that the *reduced arc costs* c' on G, where

$$c'(v, w) := c(v, w) + \pi'(v) - \pi'(w) \ , \quad \text{for each } (v, w) \in A \ , \tag{2}$$

are non-negative. It is well-known that shortest-paths problems are invariant under such a reduction of arc costs: distances μ and μ' with respect to c and c', respectively, relate to each other through $\mu'(v,w) = \mu(v,w) + \pi'(v) - \pi'(w)$, for all $v, w \in V$.

Hence, if we assume that (G,c) does not contain any negative cycles, then the all-pairs shortest-paths problem on (G,c) can be solved in time $O(nm + n^2 \log n)$. Namely, after a call of the Bellman–Ford algorithm, the all-pairs shortest-paths problem on (G,c'), with non-negative arc costs c' as in (1) and (2), can be solved by n calls of Dijkstra's algorithm, one for each source vertex. The total running time of $O(nm + n^2 \log n)$ subsumes time $\Theta(n^2)$ that is needed to compute the reduced arc costs and the actual distances in the original problem.

3 Our algorithm

To the best of our knowledge, it has not been observed before that the all-pairs shortest-paths problem on a directed graph G with n vertices, m arcs, and real arc costs c can be solved in time $O(nm + n^2 \log n)$ even if (G,c) contains negative cycles. We give an outline of the algorithm.

3.1 Strongly connected and negative components

In a first step, we compute the strongly connected components of G. For ease of presentation, we assume that we also compute the *component graph* G_{scc} of G, in which each of the n_{scc} strongly connected components of G is represented by a vertex. Since no confusion should occur, we denote both the strongly connected components of G and the vertices of G_{scc} by $C_1, \ldots, C_{n_{\mathrm{scc}}}$. The graph G_{scc} contains an arc (C_i, C_j) if and only if $i \neq j$ and $(v,w) \in A$ for some $v \in C_i, w \in C_j$. The component graph G_{scc} is an acyclic directed graph. It takes time $O(m+n)$ to compute the strongly connected components of G and the component graph G_{scc} [10]. Within the same time bound, we can ensure that the vertices of G_{scc} are *topologically sorted*, that is, the vertices are indexed so that an arc (C_i, C_j) in G_{scc} exists only if $i < j$.

Negative cycles in (G,c) can only occur inside a strongly connected component of G. Those strongly connected components that contain negative cycles (*negative components*, for short) can be detected by running the Bellman–Ford algorithm on each of the components. If for $1 \leq i \leq n_{\mathrm{scc}}$, n_i and m_i denote the number of vertices and the number of arcs, respectively, of the strongly connected component C_i, then these runs take total time $O(\sum_i n_i m_i)$, which is $O(nm)$, since $\sum_i n_i m_i \leq (\sum_i n_i)(\sum_i m_i) \leq nm$.

3.2 Infinite distances

Once we have determined the negative components of G_{scc}, it is easy to compute in a preprocessing step, for each source vertex s, the vertices $v \in V$ with $\mu(s,v) = \pm\infty$. Namely, let s be an arbitrary but fixed source vertex, which belongs to the strongly connected component C_k, for some k. In the preprocessing step with respect to s, we assign labels $M(C) \in \{\ominus_s, \odot_s, \oplus_s\}$ to the vertices C of G_{scc}. The intended meaning is that $M(C) = \ominus_s$ if, for any vertex v in C, $\mu(s,v) = -\infty$, $M(C) = \oplus_s$ if the vertices of C are not reachable from s, and $M(C) = \odot_s$ if the vertices of C have finite real distances from s. We assume the linear order $\ominus_s < \odot_s < \oplus_s$ on the set of labels. Initially, all vertices are labeled $\oplus_s$, with the only exception of C_k, which is labeled $\ominus_s$ if C_k is negative, and $\odot_s$ otherwise. We then proceed similarly to the folklore single-source shortest-paths algorithm for directed acyclic graphs; see [7]. That is, for vertices C_i, $1 \leq i \leq n_{\mathrm{scc}}$, taken in their topological order, we relax all arcs leaving C_i according to the following rules. For an arc (C_i, C_j), the label of C_j is changed only if $M(C_i) \neq \oplus_s$: if C_j is a negative component, then we set $M(C_j)$ to $\ominus_s$; otherwise, $M(C_j)$ is set to $\min\{M(C_i), M(C_j)\}$. It is easy to check that upon termination, for any vertex C of G_{scc}, its label has the intended meaning. For each source vertex s, the labeling procedure terminates in time $O(m+n)$, which amounts to a total time of $O(nm+n^2)$ spent on preprocessing.

3.3 Finite distances

We finally describe how we compute the finite distances in (G,c). We work on another auxiliary graph $\widehat{G} = (V, \widehat{A})$, where $\widehat{A}$ denotes the set A from which all arcs were removed that leave or enter vertices in negative components. Let $\widehat{c}$ denote arc costs c restricted to $\widehat{A}$. Observe that if for any pair of vertices $v, w \in V$, their distance $\mu(v,w)$ in (G,c) is not $-\infty$, then the distance of the corresponding pair of vertices in $(\widehat{G}, \widehat{c})$ equals $\mu(v,w)$. The graph $(\widehat{G}, \widehat{c})$ does not contain any negative cycles, though it may still contain arcs of negative costs. As described in Section 2, we can thus compute, in time $O(nm)$, *non-negative* reduced arc cost $\widehat{c}'$ on $\widehat{A}$ so that the shortest-paths problems on $(\widehat{G}, \widehat{c})$ and on $(\widehat{G}, \widehat{c}')$ are equivalent. The all-pairs shortest-paths problem on $(\widehat{G}, \widehat{c}')$ can then be solved by n calls of Dijkstra's algorithm, one for each source vertex. In fact, for any source vertex s, we let Dijkstra's algorithm work only on those vertices that belong to strongly connected components labeled $\odot_s$ in the corresponding preprocessing step. This suffices to compute the *finite* distances from s correctly. The total running time for these computations is $O(nm + n^2 \log n)$. This subsumes the time spent on the computations in Sections 3.1 and 3.2 as well as the time needed to compute the reduced arc costs and the actual finite distances in (G, c).

Our algorithm uses only standard algorithmic building blocks such as the computation of strongly connected components, Dijkstra's algorithm, or the Bellman–Ford algorithm, which are well-studied and easy to implement. An implementation of our algorithm will become part of the LEDA software package [8] in the near future.

Acknowledgements

The authors thank Jesper Larsson Träff and the anonymous referees for helpful comments on the presentation.

References

[1] R. Bellman, On a routing problem, Quart. Appl. Math. 16 (1958) 87–90.

[2] G. S. Brodal, Worst-case efficient priority queues, in: Proc. 7th Annual ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 52–58.

[3] E. W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1959) 269–271.

[4] J. Edmonds, R. M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, J. Assoc. Comput. Mach. 19 (1972) 248–264.

[5] L. R. Ford, Jr., D. R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, NJ, 1962.

[6] M. L. Fredman, R. E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J. Assoc. Comput. Mach. 34 (1987) 596–615.

[7] E. L. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York, 1976.

[8] K. Mehlhorn, S. Näher, LEDA. A Platform for Combinatorial and Geometric Computing, Cambridge University Press, Cambridge, 1999.

[9] G. L. Nemhauser, A generalized permanent label setting algorithm for the shortest path between specified nodes, J. Math. Anal. Appl. 38 (1972) 328–334.

[10] R. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Comput. 1 (1972) 146–160.

[11] N. Tomizawa, On some techniques useful for solution of transportation network problems, Networks 1 (1971/72) 173–194.