Open Problems

1. Is there a constant ratio approximation algorithm for the nni distance on unweighted evolutionary trees or is the $O(\log n)$-approximation the best possible?
2. Is the linear-cost subtree-transfer distance NP-hard to compute on weighted evolutionary trees if leaf labels are not allowed to be non-unique?
3. Can one improve the approximation ratio for linear-cost subtree-transfer distance on weighted evolutionary trees?

Cross References

- ► Constructing a Galled Phylogenetic Network
- ► Maximum Agreement Subtree (of 2 Binary Trees)
- ► Maximum Agreement Subtree (of 3 or More Trees)
- ► Phylogenetic Tree Construction from a Distance Matrix

Recommended Reading

1. DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J.: On the linear-cost subtree-transfer distance. Algorithmica **25**(2), 176–195 (1999)
2. DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J., Zhang, L.: On distances between phylogenetic trees, 8th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 427–436 (1997)
3. DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J., Wang, L., Zhang, L.: Computing Distances between Evolutionary Trees. In: Du, D.Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization. Kluwer Academic Publishers, Norwell, **2**, 35–76 (1998)
4. DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J., Zhang, L.: On Computing the Nearest Neighbor Interchange Distance. In: Du, D.Z., Pardalos, P.M., Wang, J. (eds.) Proceedings of the DIMACS Workshop on Discrete Problems with Medical Applications, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. Am. Math. Soc. **55**, 125–143 (2000)
5. Hein, J.: Reconstructing evolution of sequences subject to recombination using parsimony. Math. Biosci. **98**, 185–200 (1990)
6. Hein, J.: A heuristic method to reconstruct the history of sequences subject to recombination. J. Mol. Evol. **36**, 396–405 (1993)
7. Hein, J., Jiang, T., Wang, L., Zhang, K.: On the complexity of comparing evolutionary trees. Discret. Appl. Math. **71**, 153–169 (1996)
8. Kuhner, M., Felsenstein, J.: A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. Mol. Biol. Evol. **11**(3), 459–468 (1994)
9. Moore, G.W., Goodman, M., Barnabas, J.: An iterative approach from the standpoint of the additive hypothesis to the dendrogram problem posed by molecular data sets. J. Theor. Biol. **38**, 423–457 (1973)
10. Robinson, D.F.: Comparison of labeled trees with valency three. J Combinator. Theory Series B **11**, 105–119 (1971)

Negative Cycles in Weighted Digraphs

1994; Kavvadias, Pantziou, Spirakis, Zaroliagis

Christos Zaroliagis
Computer Engineering & Informatics,
University of Patras, Patras, Greece

Problem Definition

Let $G = (V, E)$ be an n-vertex, m-edge directed graph (digraph), whose edges are associated with a real-valued cost function $wt : E \to \mathbb{R}$. The cost, $wt(P)$, of a path P in G is the sum of the costs of the edges of P. A simple path C whose starting and ending vertices coincide is called a cycle. If $wt(C) < 0$, then C is called a *negative cycle*. The goal of the negative cycle problem is to detect whether there is such a cycle in a given digraph G with real-valued edge costs, and if indeed exists to output the cycle.

The negative cycle problem is closely related to the shortest path problem. In the latter, a minimum cost path between two vertices s and t is sought. It is easy to see that an s-t shortest path exists if and only if no s-t path in G contains a negative cycle [1,13]. It is also well-known that shortest paths from a given vertex s to all other vertices form a tree called *shortest path tree* [1,13].

Key Results

For the case of general digraphs, the best algorithm to solve the negative cycle problem (or to compute the shortest path tree, if such a cycle does not exist) is the classical Bellman–Ford algorithm that takes $O(nm)$ time (see e. g., [1]). Alternative methods with the same time complexity are given in [4,7,12,13]. Moreover, in [11, Chap. 7] an extension of the Bellman–Ford algorithm is described which, in addition to detecting and reporting the existing negative cycles (if any), builds a shortest path tree rooted a some vertex s reaching those vertices u whose shortest s-u path does not contain a negative cycle. If edge costs are integers larger than $-L$ ($L \geq 2$), then a better algorithm was given in [6] that runs in $O(m\sqrt{n}\log L)$ time, and it is based on bit scaling.

A simple deterministic algorithm that runs in $O(n^2 \log n)$ expected time with high probability is given in [10] for a large class of input distributions, where the edge costs are chosen randomly according to the endpoint-independent model (this model includes the common case where all edge costs are chosen independently from the same distribution).

Better results are known for several important classes of sparse digraphs (i. e., digraphs with $m = O(n)$ edges) such as planar digraphs, outerplanar digraphs, digraphs of small genus, and digraphs of small treewidth.

For general sparse digraphs, an algorithm is given in [8] that solves the negative cycle problem in $O(n + \tilde{\gamma}^{1.5} \log \tilde{\gamma})$ time, where $\tilde{\gamma}$ is a topological measure of the input sparse digraph G, and whose value varies from 1 up to $\Theta(n)$. Informally, $\tilde{\gamma}$ represents the minimum number of outerplanar subgraphs, satisfying certain separation properties, into which G can be decomposed. In particular, $\tilde{\gamma}$ is proportional to $\gamma(G) + q$, where G is supposed to be embedded into an orientable surface of genus $\gamma(G)$ so as to minimize the number q of faces that collectively cover all vertices. For instance, if G is outerplanar, then $\tilde{\gamma} = 1$, which implies an optimal $O(n)$ time algorithm for this case. The algorithm in [8] does not require such an embedding to be provided by the input. In the same paper, it is shown that random $G_{n,p}$ graphs with threshold function $1/n$ are planar with probability one and have an expected value for $\tilde{\gamma}$ equal to $O(1)$. Furthermore, an efficient parallelization of the algorithm on the CREW PRAM model of computation is provided in [8].

Better bounds for planar digraphs are as follows. If edge costs are integers, then an algorithm running in $O(n^{4/3} \log(nL))$ time is given in [9]. For real edge costs, an $O(n \log^3 n)$-time algorithm was given in [5].

An optimal $O(n)$-time algorithm is given in [3] for the case of digraphs with small treewidth (and real edge costs). Informally, the treewidth t of a graph G is a parameter which measures how close is the structure of G to a tree. For instance, the class of graphs of small treewidth includes series-parallel graphs ($t = 2$) and outerplanar graphs ($t = 2$). An optimal parallel algorithm for the same problem, on the EREW PRAM model of computation, is provided in [2].

Applications

Finding negative cycles in a digraph is a fundamental combinatorial and network optimization problem that spans a wide range of applications including: shortest path computation, two dimensional package element, minimum cost flows, minimal cost-to-time ratio, model verification, compiler construction, software engineering, VLSI design, scheduling, circuit production, constraint programming and image processing. For instance, the isolation of negative feedback loops is imperative in the design of VLSI circuits. It turns out that such loops correspond to negative cost cycles in the so-called amplifier-gain graph of the circuit. In constraint programming, it is required to check the feasibility of sets of constraints. Systems of difference constraints can be represented by constraint graphs, and one can show that such a system is feasible if and only if there are no negative cost cycles in its corresponding constraint graph. In zero-clairvoyant scheduling, the problem of checking whether there is a valid schedule in such a scheduling system can be reduced to detecting negative cycles in an appropriately defined graph. For further discussion on these and other applications see [1,12,14].

Open Problems

The negative cycle problem is closely related to the shortest path problem. The existence of negative edge costs makes the solution of the negative cycle problem or the computation of a shortest path tree more difficult and thus more time consuming compared to the time required to solve the shortest path tree problem in digraphs with non-negative edge costs. For instance, for digraphs with real edge costs, compare the $O(nm)$-time algorithm in the former case with the $O(m + n \log n)$-time algorithm for the latter case (Dijkstra's algorithm implemented with an efficient priority queue; see e. g., [1]).

It would therefore be interesting to try to reduce the gap between the above two time complexities, even for special classes of graphs or the case of integer costs.

The only case where these two complexities coincide concerns the digraphs of small treewidth [3], making it the currently most general such class of graphs. For planar digraphs, the result in [5] is only a polylogarithmic factor away from the $O(n)$-time algorithm in [9] that computes a shortest path tree when the edge costs are non-negative.

Experimental Results

An experimental study for the negative cycle problem is conducted in [4]. In that paper, several methods that combine a shortest path algorithm (based on the Bellman–Ford approach) with a cycle detection strategy are investigated, along with some new variations of them. It turned out that the performance of algorithms for the negative cycle problem depends on the number and the size of the negative cycles. This gives rise to a collection of problem families for testing negative cycle algorithms.

A follow-up of the above study is presented in [14], where two new heuristics are introduced and are incorporated on three of the algorithms considered in [4] (the original Bellman–Ford and the variations in [13] and [7]), achieving dramatic improvements. The data sets considered in [14] are those in [4].

Data Sets

Data set generators and problem families are described in [4], and are available from http://www.avglab.com/andrew/soft.html.

URL to Code

The code used in [4] is available from http://www.avglab.com/andrew/soft.html.

Cross References

- ► All Pairs Shortest Paths in Sparse Graphs
- ► All Pairs Shortest Paths via Matrix Multiplication
- ► Single-Source Shortest Paths

Recommended Reading

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows. Prentice-Hall, Englewood Cliffs (1993)
2. Chaudhuri, S., Zaroliagis, C.: Shortest Paths in Digraphs of Small Treewidth. Part II: Optimal Parallel Algorithms. Theor. Comput. Sci. **203**(2), pp. 205–223 (1998)
3. Chaudhuri, S., Zaroliagis, C.: Shortest Paths in Digraphs of Small Treewidth. Part I: Sequential Algorithms. Algorithmca **27**(3), pp. 212–226 (2000)
4. Cherkassky, B.V., Goldberg, A.V.: Negative-Cycle Detection Algorithms. Math. Program. **85**, pp. 277–311 (1999)
5. Fakcharoenphol, J., Rao, S.: Planar Graphs, Negative Weight Edges, Shortest Paths, and near Linear Time. In: Proc. 42nd IEEE Symp. on Foundations of Computer Science – FOCS (2001), pp. 232–241. IEEE Computer Society Press, Los Alamitos (2001)
6. Goldberg, A.V.: Scaling Algorithms for the Shortest Paths Problem. SIAM J. Comput. **24**, pp. 494–504 (1995)
7. Goldberg, A.V., Radzik, T.: A Heuristic Improvement of the Bellman—Ford Algorithm. Appl. Math. Lett. **6**(3), pp. 3–6 (1993)
8. Kavvadias, D., Pantziou, G., Spirakis, P., Zaroliagis, C.: Efficient Sequential and Parallel Algorithms for the Negative Cycle Problem. In: Algorithms and Computation – ISAAC'94. Lect. Notes Comput. Sci., vol. 834, pp.270–278. Springer, Heidelberg (1994)
9. Klein, P., Rao, S., Rauch, M., Subramanian, S.: Faster shortest-path algorithms for planar graphs. J. Comput. Syst. Sci. **5**(1), pp. 3–23 (1997)
10. Kolliopoulos, S.G., Stein, C.: Finding Real-Valued Single-Source Shortest Paths in $o(n^3)$ Expected Time. J. Algorithms **28**, pp. 125–141 (1998)
11. Mehlhorn, K., Näher, S.: LEDA: A Platform for Combinatorial and Geometric Computing. Cambridge University Press, Cambridge (1999)
12. Spirakis, P., Tsakalidis, A.: A Very Fast, Practical Algorithm for Finding a Negative Cycle in a Digraph. In Proc. of 13th ICALP, pp. 397–406 (1986)
13. Tarjan, R.E.: Data Structures and Network Algorithms. SIAM, Philadelphia (1983)
14. Wong, C.H., Tam, Y.C.: Negative Cycle Detection Problem. In: Algorithms – ESA 2005. Lecture Notes in Computer Science, vol. 3669, pp. 652–663. Springer, Heidelberg (2005)

Non-approximability of Bimatrix Nash Equilibria

2006; Chen, Deng, Teng

Xi Chen[1], Xiaotie Deng[2]

[1] Computer Science and Technology, Tsinghua University, Beijing, Beijing, China

[2] Department of Computer Science, City University of Hong Kong, Hong Kong, China

Keywords and Synonyms

Approximate Nash equilibrium

Problem Definition

In this entry, the following two problems are considered: 1) the problem of finding an approximate Nash equilibrium in a positively normalized bimatrix (or two-player) game; and 2) the smoothed complexity of finding an exact Nash equilibrium in a bimatrix game. It turns out that these two problems are strongly correlated [3].

Let $\mathcal{G} = (\mathbf{A}, \mathbf{B})$ be a bimatrix game, where $\mathbf{A} = (a_{i,j})$ and $\mathbf{B} = (b_{i,j})$ are both $n \times n$ matrices. Game $\mathcal{G}$ is said to be positively normalized, if $0 \le a_{i,j}, b_{i,j} \le 1$ for all $1 \le i, j \le n$.

Let $\mathbb{P}^n$ denote the set of all probability vectors in $\mathbb{R}^n$, i. e., non-negative vectors whose entries sum to 1. A Nash equilibrium [8] of $\mathcal{G} = (\mathbf{A}, \mathbf{B})$ is a pair of mixed strategies $(\mathbf{x}^* \in \mathbb{P}^n, \mathbf{y}^* \in \mathbb{P}^n)$ such that for all $\mathbf{x}, \mathbf{y} \in \mathbb{P}^n$,

$$(\mathbf{x}^*)^{\mathrm{T}}\mathbf{A}\mathbf{y}^* \ge \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{y}^* \quad \text{and} \quad (\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}\mathbf{y}^* \ge (\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}\mathbf{y}\,,$$

while an ϵ-approximate Nash equilibrium is a pair $(\mathbf{x}^* \in \mathbb{P}^n, \mathbf{y}^* \in \mathbb{P}^n)$ that satisfies

$$(\mathbf{x}^*)^{\mathrm{T}}\mathbf{A}\mathbf{y}^* \ge \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{y}^* - \epsilon \quad \text{and}$$
$$(\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}\mathbf{y}^* \ge (\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}\mathbf{y} - \epsilon\,, \quad \text{for all} \quad \mathbf{x}, \mathbf{y} \in \mathbb{P}^n\,.$$

In the smoothed analysis [11] of bimatrix games, a perturbation of magnitude $\sigma > 0$ is first applied to the input game: For a positively normalized $n \times n$ game $\mathcal{G} = (\overline{\mathbf{A}}, \overline{\mathbf{B}})$, let $\mathbf{A}$ and $\mathbf{B}$ be two matrices with

$$a_{i,j} = \overline{a}_{i,j} + r^A_{i,j} \quad \text{and} \quad b_{i,j} = \overline{b}_{i,j} + r^B_{i,j}, \quad \forall 1 \le i, j \le n,$$

while $r^A_{i,j}$ and $r^B_{i,j}$ are chosen independently and uniformly from interval $[-\sigma, \sigma]$ or from Gaussian distribution with variance σ^2. These two kinds of perturbations are referred to as *σ-uniform* and *σ-Gaussian perturbations*, respectively. An algorithm for bimatrix games has *polynomial smoothed complexity* (under σ-uniform or σ-Gaussian perturbations) [11], if it finds a Nash equilibrium of game $(\mathbf{A}, \mathbf{B})$ in expected time poly$(n, 1/\sigma)$, for all $(\overline{\mathbf{A}}, \overline{\mathbf{B}})$.