F

Facility Location

1997; Shmoys, Tardos, Aardal

Karen Aardal[1,2], Jaroslaw Byrka[1,2], Mohammad Mahdian[3]
[1] CWI, Amsterdam, The Netherlands
[2] Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands
[3] Yahoo! Research, Santa Clara, CA, USA

Keywords and Synonyms

Plant location; Warehouse location

Problem Definition

Facility location problems concern situations where a planner needs to determine the location of facilities intended to serve a given set of clients. The objective is usually to minimize the sum of the cost of opening the facilities and the cost of serving the clients by the facilities, subject to various constraints, such as the number and the type of clients a facility can serve. There are many variants of the facility location problem, depending on the structure of the cost function and the constraints imposed on the solution. Early references on facility location problems include Kuehn and Hamburger [35], Balinski and Wolfe [8], Manne [40], and Balinski [7]. Review works include Krarup and Pruzan [34] and Mirchandani and Francis [42]. It is interesting to notice that the algorithm that is probably one of the most effective ones to solve the uncapacitated facility location problem to optimality is the primal-dual algorithm combined with branch-and-bound due to Erlenkotter [16] dating back to 1978. His primal-dual scheme is similar to techniques used in the modern literature on approximation algorithms.

More recently, extensive research into approximation algorithms for facility location problems has been carried out. Review articles on this topic include Shmoys [49,50] and Vygen [55]. Besides its theoretical and practical importance, facility location problems provide a showcase of common techniques in the field of approximation algorithms, as many of these techniques such as linear programming rounding, primal-dual methods, and local search have been applied successfully to this family of problems. This entry defines several facility location problems, gives a few historical pointers, and lists approximation algorithms with an emphasis on the results derived in the paper by Shmoys, Tardos, and Aardal [51]. The techniques applied to the *uncapacitated facility location* (UFL) problem are discussed in some more detail.

In the UFL problem, a set $\mathcal{F}$ of n_f *facilities* and a set C of n_c *clients* (also known as *cities*, or *demand points*) are given. For every facility $i \in \mathcal{F}$, the *facility opening* cost is equal to f_i. Furthermore, for every facility $i \in \mathcal{F}$ and client $j \in C$, there is a *connection cost* c_{ij}. The objective is to open a subset of the facilities and connect each client to an open facility so that the total cost is minimized. Notice that once the set of open facilities is specified, it is optimal to connect each client to the open facility that yields smallest connection cost. Therefore, the objective is to find a set $S \subseteq \mathcal{F}$ that minimizes $\sum_{i \in S} f_i + \sum_{j \in C} \min_{i \in S}\{c_{ij}\}$. This definition and the definitions of other variants of the facility location problem in this entry assume unit demand at each client. It is straightforward to generalize these definitions to the case where each client has a given demand. The UFL problem can be formulated as the following integer program due to Balinski [7]. Let $y_i,\ i \in \mathcal{F}$ be equal to 1 if facility i is open, and equal to 0 otherwise. Let $x_{ij},\ i \in \mathcal{F},\ j \in C$ be the fraction of client j assigned to facility i.

$$\min \sum_{i \in \mathcal{F}} f_i y_i + \sum_{i \in \mathcal{F}} \sum_{j \in C} c_{ij} x_{ij} \tag{1}$$

$$\text{subject to} \sum_{i \in \mathcal{F}} x_{ij} = 1, \quad \text{for all } j \in C, \tag{2}$$

$$x_{ij} - y_i \leq 0, \quad \text{for all } i \in \mathcal{F},\ j \in C \tag{3}$$

$$x \geq 0,\ y \in \{0,1\}^{n_f} \tag{4}$$

In the *linear programming (LP) relaxation* of UFL the constraint $y \in \{0,1\}^{n_f}$ is substituted by the constraint $y \in [0,1]^{n_f}$. Notice that in the uncapacitated case, it is not necessary to require $x_{ij} \in \{0,1\},\ i \in \mathcal{F},\ j \in C$ if each client has to be serviced by precisely one facility, as $0 \le x_{ij} \le 1$ by constraints (2) and (4). Moreover, if x_{ij} is not integer, then it is always possible to create an integer solution with the same cost by assigning client j completely to one of the facilities currently servicing j.

A γ-approximation algorithm is a polynomial algorithm that, in case of minimization, is guaranteed to produce a feasible solution having value *at most* γz^*, where z^* is the value of an optimal solution, and $\gamma \ge 1$. If $\gamma = 1$ the algorithm produces an optimal solution. In case of maximization, the algorithm produces a solution having value *at least* γz^*, where $0 \le \gamma \le 1$.

Hochbaum [25] developed an $O(\log n)$-approximation algorithm for UFL. By a straightforward reduction from the Set Cover problem, it can be shown that this cannot be improved unless $NP \subseteq DTIME[n^{O(\log\log n)}]$ due to a result by Feige [17]. However, if the connection costs are restricted to come from distances in a metric space, namely $c_{ij} = c_{ji} \ge 0$ for all $i \in \mathcal{F}, j \in C$ (nonnegativity and symmetry) and $c_{ij} + c_{ji'} + c_{i'j'} \ge c_{ij'}$ for all $i, i' \in \mathcal{F}, j, j' \in C$ (triangle inequality), then constant approximation guarantees can be obtained. In all results mentioned below, except for the maximization objectives, it is assumed that the costs satisfy these restrictions. If the distances between facilities and clients are Euclidean, then for some location problems approximation schemes have been obtained [5].

Variants and Related Problems

A variant of the uncapacitated facility location problem is obtained by considering the objective coefficients c_{ij} as the per unit profit of servicing client j from facility i. The *maximization version* of UFL, max-UFL is obtained by maximizing the profit minus the facility opening cost, i. e., $\max \sum_{i\in\mathcal{F}} \sum_{j\in C} c_{ij}x_{ij} - \sum_{i\in\mathcal{F}} f_i y_i$. This variant was introduced by Cornuéjols, Fisher, and Nemhauser [15].

In the *k-median problem* the facility opening cost is removed from the objective function (1) to obtain $\min \sum_{i\in M} \sum_{j\in N} c_{ij}x_{ij}$, and the constraint that no more than k facilities may be opened, $\sum_{i\in M} y_i \le k$, is added. In the *k-center problem* the constraint $\sum_{i\in M} y_i \le k$ is again included, and the objective function here is to minimize the maximum distance used on a link between an open facility and a client.

In the *capacitated facility location problem* a capacity constraint $\sum_{j\in C} x_{ij} \le u_i y_i$ is added for all $i \in \mathcal{F}$. Here it is important to distinguish between the *splittable* and the *unsplittable* case, and also between *hard capacities* and *soft capacities*. In the splittable case one has $x \ge 0$, allowing for a client to be serviced by multiple depots, and in the unsplittable case one requires $x \in \{0,1\}^{n_f \times n_c}$. If each facility can be opened at most once (i. e., $y_i \in \{0,1\}$), the capacities are called hard; otherwise, if the problem allows a facility i to be opened any number r of times to serve ru_i clients, the capacities are called soft.

In the k-level facility location problem, the following are given: a set C of clients, k disjoint sets $\mathcal{F}_1, \ldots, \mathcal{F}_k$ of facilities, an opening cost for each facility, and connection costs between clients and facilities. The goal is to connect each client j through a path $i_1,\ldots,i_k$ of open facilities, with $i_\ell \in \mathcal{F}_\ell$. The connection cost for this client is $c_{ji_1} + c_{i_1 i_2} + \cdots + c_{i_{k-1} i_k}$. The goal is to minimize the sum of connection costs and facility opening costs.

The problems mentioned above have all been considered by Shmoys, Tardos, and Aardal [51], with the exceptions of max-UFL, and the k-center and k-median problems. The max-UFL variant is included for historical reasons, and k-center and k-median are included since they have a rich history and since they are closely related to UFL. Results on the capacitated facility location problem with hard capacities are mentioned as this, at least from the application point of view, is a more realistic model than the soft capacity version, which was treated in [51]. For k-level facility location, Shmoys et al. considered the case $k = 2$. Here, the problem for general k is considered.

There are many other variants of the facility location problem that are not discussed here. Examples include K-facility location [33], universal facility location [24,38], online facility location [3,18,41], fault tolerant facility location [28,30,54], facility location with outliers [12,28], multicommodity facility location [48], priority facility location [37,48], facility location with hierarchical facility costs [52], stochastic facility location [23,37,46], connected facility location [53], load-balanced facility location [22,32,37], concave-cost facility location [24], and capacitated-cable facility location [37,47].

Key Results

Many algorithms have been proposed for location problems. To begin with, a brief description of the algorithms of Shmoys, Tardos, and Aardal [51] is given. Then, a quick overview of some key results is presented. Some of the algorithms giving the best values of the approximation guarantee γ are based on solving the LP-relaxation by a polynomial algorithm, which can actually be quite time consuming, whereas some authors have suggested fast combi-

natorial algorithms for facility location problems with less competitive γ-values. Due to space restrictions the focus of this entry is on the algorithms that yield the best approximation guarantees. For more references the survey papers by Shmoys [49,50] and by Vygen [55] are recommended.

The Algorithms of Shmoys, Tardos, and Aardal

First the algorithm for UFL is described, and then the results that can be obtained by adaptations of the algorithm to other problems are mentioned.

The algorithm solves the LP relaxation and then, in two stages, modifies the obtained fractional solution. The first stage is called *filtering* and it is designed to bound the connection cost of each client to the most distant facility fractionally serving him. To do so, the facility opening variables y_i are scaled up by a constant and then the connection variables x_{ij} are adjusted to use the closest possible facilities.

To describe the second stage, the notion of *clustering*, formalized later by Chudak and Shmoys [13] is used. Based on the fractional solution, the instance is cut into pieces called *clusters*. Each cluster has a distinct client called the *cluster center*. This is done by iteratively choosing a client, not covered by the previous clusters, as the next cluster center, and adding to this cluster the facilities that serve the cluster center in the fractional solution, along with other clients served by these facilities. This construction of clusters guarantees that the facilities in each cluster are open to a total extent of one, and therefore after opening the facility with the smallest opening cost in each cluster, the total facility opening cost that is paid does not exceed the facility opening cost of the fractional solution. Moreover, by choosing clients for the cluster centers in a greedy fashion, the algorithm makes each cluster center the minimizer of a certain cost function among the clients in the cluster. The remaining clients in the cluster are also connected to the opened facility. The triangle inequality for connection costs is now used to bound the cost of this connection. For UFL, this filtering and rounding algorithm is a 4-approximation algorithm. Shmoys et al. also show that if the filtering step is substituted by *randomized filtering*, an approximation guarantee of 3.16 is obtained.

In the same paper, adaptations of the algorithm, with and without randomized filtering, was made to yield approximation algorithms for the soft-capacitated facility location problem, and for the 2-level uncapacitated problem. Here, the results obtained using randomized filtering are discussed.

For the problem with soft capacities two versions of the problem were considered. Both have equal capacities, i. e., $u_i = u$ for all $i \in \mathcal{F}$. In the first version, a solution is "feasible" if the y-variables either take value 0, or a value between 1 and $\gamma' \geq 1$. Note that γ' is not required to be integer, so the constructed solution is not necessarily integer. This can be interpreted as allowing for each facility i to expand to have capacity $\gamma' u$ at a cost of $\gamma' f_i$. A (γ, γ')-approximation algorithm is a polynomial algorithm that produces such a feasible solution having a total cost within a factor of γ of the true optimal cost, i. e., with $y \in \{0, 1\}^{n_f}$. Shmoys et al. developed a $(5.69, 4.24)$-approximation algorithm for the splittable case of this problem, and a $(7.62, 4.29)$-approximation algorithm for the unsplittable case.

In the second soft-capacitated model, the original problem is changed to allow for the y-variables to take nonnegative integer values, which can be interpreted as allowing multiple facilities of capacity u to be opened at each location. The approximation algorithms in this case produces a solution that is feasible with respect to this modified model. It is easy to show that the approximation guarantees obtained for the previous model also hold in this case, i. e., Shmoys et al. obtained a 5.69-approximation algorithm for splittable demands and a 7.62-approximation algorithm for unsplittable demands. This latter model is the one considered in most later papers, so this is the model that is referred to in the paragraph on soft capacity results below.

UFL

The first algorithm with constant performance guarantee was the 3.16-approximation algorithm by Shmoys, Tardos, and Aardal, see above. Since then numerous improvements have been made. Guha and Khuller [19,20] proved a lower bound on approximability of 1.463, and introduced a *greedy augmentation procedure*. A series of approximation algorithms based on LP-rounding was then developed (see e. g. [10,13]). There are also greedy algorithms that only use the LP-relaxation implicitly to obtain a lower bound for a primal-dual analysis. An example is the JMS 1.61-approximation algorithm developed by Jain, Mahdian, and Saberi [29]. Some algorithms combine several techniques, like the 1.52-approximation algorithm of Mahdian, Ye, and Zhang [39], which uses the JMS algorithm and the greedy augmentation procedure. Currently, the best known approximation guarantee is 1.5 reported by Byrka [10]. It is obtained by combining a randomized LP-rounding algorithm with the greedy JMS algorithm.

max-UFL

The first constant factor approximation algorithm was derived in 1977 by Cornuéjols et al. [15] for max-UFL. They showed that opening one facility at a time in a greedy fashion, choosing the facility to open as the one with highest marginal profit, until no facility with positive marginal profit can be found, yields a $(1 - 1/e) \approx 0.632$-approximation algorithm. The current best approximation factor is 0.828 by Ageev and Sviridenko [2].

k-median, *k*-center

The first constant factor approximation algorithm for the *k*-median problem is due to Charikar, Guha, Tardos, and Shmoys [11]. This LP-rounding algorithm has the approximation ratio of $6\frac{2}{3}$. The currently best known approximation ratio is $3 + \epsilon$ achieved by a local search heuristic of Arya, et al. [6] (see also a separate entry *k-median and Facility Location*).

The first constant factor approximation algorithm for the *k*-center problem was given by Hochbaum and Shmoys [26], who developed a 2-approximation algorithm. This performance guarantee is the best possible unless $P = NP$.

Capacitated Facility Location

For the soft-capacitated problem with equal capacities, the first constant factor approximation algorithms are due to Shmoys et al. [51] for both the splittable and unsplittable demand cases, see above. Recently, a 2-approximation algorithm for the soft capacitated facility location problem with unsplittable unit demands was proposed by Mahdian et al. [39]. The integrality gap of the LP relaxation for the problem is also 2. Hence, to improve the approximation guarantee one would have to develop a better lower bound on the optimal solution.

In the hard capacities version it is important to allow for splitting the demands, as otherwise even the feasibility problem becomes difficult. Suppose demands are splittable, then we may to distinguish between the equal capacity case, where $u_i = u$ for all $i \in \mathcal{F}$, and the general case. For the problem with equal capacities, a 5.83-approximation algorithm was given by Chudak and Wiliamson [14]. The first constant factor approximation algorithm, with $\gamma = 8.53 + \epsilon$, for general capacities was given by Pál, Tardos, and Wexler [44]. This was later improved by Zhang, Chen, and Ye [57] who obtained a 5.83-approximation algorithm also for general capacities.

k-level Problem

The first constant factor approximation algorithm for $k = 2$ is due to Shmoys et al. [51], with $\gamma = 3.16$. For general k, the first algorithm, having $\gamma = 3$, was proposed by Aardal, Chudak, and Shmoys [1]. For $k = 2$, Zhang [56] developed a 1.77-approximation algorithm. He also showed that the problem for $k = 3$ and $k = 4$ can be approximated by $\gamma = 2.523$ [1] and $\gamma = 2.81$ respectively.

Applications

Facility location has numerous applications in the field of operations research. See the book edited by Mirchandani and Francis [42] or the book by Nemhauser and Wolsey [43] for a survey and a description of applications of facility location in problems such as plant location and locating bank accounts. Recently, the problem has found new applications in network design problems such as placement of routers and caches [22,36], agglomeration of traffic or data [4,21], and web server replications in a content distribution network [31,45].

Open Problems

A major open question is to determine the exact approximability threshold of UFL and close the gap between the upper bound of 1.5 [10] and the lower bound of 1.463 [20]. Another important question is to find better approximation algorithms for *k*-median. In particular, it would be interesting to find an LP-based 2-approximation algorithm for *k*-median. Such an algorithm would determine the integrality gap of the natural LP relaxation of this problem, as there are simple examples that show that this gap is at least 2.

Experimental Results

Jain et al. [28] published experimental results comparing various primal-dual algorithms. A more comprehensive experimental study of several primal-dual, local search, and heuristic algorithms is performed by Hoefer [27]. A collection of data sets for UFL and several other location problems can be found in the OR-library maintained by Beasley [9].

Cross References

- ▶ Assignment Problem
- ▶ Bin Packing (hardness of Capacitated Facility Location with unsplittable demands)

[1] This value of γ deviates slightly from the value 2.51 given in the paper. The original argument contained a minor calculation error.

- ▶ Circuit Placement
- ▶ Greedy Set-Cover Algorithms (hardness of a variant of UFL, where facilities may be built at all locations with the same cost)
- ▶ Local Approximation of Covering and Packing Problems
- ▶ Local Search for K-medians and Facility Location

Recommended Reading

1. Aardal, K., Chudak, F.A., Shmoys, D.B.: A 3-approximation algorithm for the k-level uncapacitated facility location problem. Inf. Process. Lett. **72**, 161–167 (1999)
2. Ageev, A.A., Sviridenko, M.I.: An 0.828-approximation algorithm for the uncapacitated facility location problem. Discret. Appl. Math. **93**, 149–156 (1999)
3. Anagnostopoulos, A., Bent, R., Upfal, E., van Hentenryck, P.: A simple and deterministic competitive algorithm for online facility location. Inf. Comput. **194**(2), 175–202 (2004)
4. Andrews, M., Zhang, L.: The access network design problem. In: Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 40–49. IEEE Computer Society, Los Alamitos, CA, USA (1998)
5. Arora, S., Raghavan, P., Rao, S.: Approximation schemes for Euclidean k-medians and related problems. In: Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC), pp. 106–113. ACM, New York (1998)
6. Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., Pandit, V.: Local search heuristics for k-median and facility location problems. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 21–29. ACM, New York (2001)
7. Balinski, M.L.: On finding integer solutions to linear programs. In: Proceedings of the IBM Scientific Computing Symposium on Combinatorial Problems, pp. 225–248 IBM, White Plains, NY (1966)
8. Balinski, M.L., Wolfe, P.: On Benders decomposition and a plant location problem. In ARO-27. Mathematica Inc. Princeton (1963)
9. Beasley, J.E.: Operations research library. http://people.brunel.ac.uk/~mastjjb/jeb/info.html. Accessed 2008
10. Byrka, J.: An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. In: Proceedings of the 10th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), Lecture Notes in Computer Science, vol. 4627, pp. 29–43. Springer, Berlin (2007)
11. Charikar, M., Guha, S., Tardos, E., Shmoys, D.B.: A constant-factor approximation algorithm for the k-median problem. In: Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC), pp. 1–10. ACM, New York (1999)
12. Charikar, M., Khuller, S., Mount, D., Narasimhan, G.: Facility location with outliers. In: Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 642–651. SIAM, Philadelphia (2001)
13. Chudak, F.A., Shmoys, D.B.: Improved approximation algorithms for the uncapacitated facility location problem. SIAM J Comput. **33**(1), 1–25 (2003)
14. Chudak, F.A., Wiliamson, D.P.: Improved approximation algorithms for capacitated facility location problems. In: Proceedings of the 7th Conference on Integer Programing and Combinatorial Optimization (IPCO). Lecture Notes in Computer Science, vol. 1610, pp. 99–113. Springer, Berlin (1999)
15. Cornuéjols, G., Fisher, M.L., Nemhauser, G.L.: Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. Manag. Sci. **8**, 789–810 (1977)
16. Erlenkotter, D.: A dual-based procedure for uncapacitated facility location problems. Oper. Res. **26**, 992–1009 (1978)
17. Feige, U.: A threshold of $\ln n$ for approximating set cover. J. ACM **45**, 634–652 (1998)
18. Fotakis, D.: On the competitive ratio for online facility location. In: Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science, vol. 2719, pp. 637–652. Springer, Berlin (2003)
19. Guha, S., Khuller, S.: Greedy strikes back: Improved facility location algorithms. In: Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 228–248. SIAM, Philadelphia (1998)
20. Guha, S., Khuller, S.: Greedy strikes back: Improved facility location algorithms. J. Algorithms **31**, 228–248 (1999)
21. Guha, S., Meyerson, A., Munagala, K.: A constant factor approximation for the single sink edge installation problem. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 383–388. ACM Press, New York (2001)
22. Guha, S., Meyerson, A., Munagala, K.: Hierarchical placement and network design problems. In: Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 603–612. IEEE Computer Society, Los Alamitos, CA, USA (2000)
23. Gupta, A., Pál, M., Ravi, R., Sinha, A.: Boosted sampling: approximation algorithms for stochastic optimization. In: Proceedings of the 36st Annual ACM Symposium on Theory of Computing (STOC), pp. 417–426. ACM, New York (2004)
24. Hajiaghayi, M., Mahdian, M., Mirrokni, V.S.: The facility location problem with general cost functions. Netw. **42**(1), 42–47 (2003)
25. Hochbaum, D.S.: Heuristics for the fixed cost median problem. Math. Program. **22**(2), 148–162 (1982)
26. Hochbaum, D.S., Shmoys, D.B.: A best possible approximation algorithm for the k-center problem. Math. Oper. Res. **10**, 180–184 (1985)
27. Hoefer, M.: Experimental comparison of heuristic and approximation algorithms for uncapacitated facility location. In: Proceedings of the 2nd International Workshop on Experimental and Efficient Algorithms (WEA). Lecture Notes in Computer Science, vol. 2647, pp. 165–178. Springer, Berlin (2003)
28. Jain, K., Mahdian, M., Markakis, E., Saberi, A., Vazirani, V.V.: Approximation algorithms for facility location via dual fitting with factor-revealing LP. J. ACM **50**(6), 795–824 (2003)
29. Jain, K., Mahdian, M., Saberi, A.: A new greedy approach for facility location problems. In: Proceedings of the 34st Annual ACM Symposium on Theory of Computing (STOC) pp. 731–740, ACM Press, New York (2002)
30. Jain, K., Vazirani, V.V.: An approximation algorithm for the fault tolerant metric facility location problem. In: Approximation Algorithms for Combinatorial Optimization, Proceedings of APPROX (2000), vol. (1913) of Lecture Notes in Computer Science, pp. 177–183. Springer, Berlin (2000)
31. Jamin, S., Jin, C., Jin, Y., Raz, D., Shavitt, Y., Zhang, L.: On the placement of internet instrumentations. In: Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Com-

munications Societies (INFOCOM), vol. 1, pp. 295–304. IEEE Computer Society, Los Alamitos, CA, USA (2000)
32. Karger, D., Minkoff, M.: Building Steiner trees with incomplete global knowledge. In: Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, pp. 613–623. Los Alamitos (2000)
33. Krarup, J., Pruzan, P.M.: Ingredients of locational analysis. In: Mirchandani, P., Francis, R. (eds.) Discrete Location Theory, pp. 1–54. Wiley, New York (1990)
34. Krarup, J., Pruzan, P.M.: The simple plant location problem: Survey and synthesis. Eur. J. Oper. Res. **12**, 38–81 (1983)
35. Kuehn, A.A., Hamburger, M.J.: A heuristic program for locating warehouses. Manag. Sci. **9**, 643–666 (1963)
36. Li, B., Golin, M., Italiano, G., Deng, X., Sohraby, K.: On the optimal placement of web proxies in the internet. In: Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 1282–1290. IEEE Computer Society, Los Alamitos (1999)
37. Mahdian, M.: Facility Location and the Analysis of Algorithms through Factor-Revealing Programs. Ph. D. thesis, MIT, Cambridge (2004)
38. Mahdian, M., Pál, M.: Universal facility location. In: Proceedings of the 11th Annual European Symposium on Algorithms (ESA). Lecture Notes in Computer Science, vol. 2832, pp. 409–421. Springer, Berlin (2003)
39. Mahdian, M., Ye, Y., Zhang, J.: Approximation algorithms for metric facility location problems. SIAM J. Comput. **36**(2), 411–432 (2006)
40. Manne, A.S.: Plant location under economies-of-scale – decentralization and computation. Manag. Sci. **11**, 213–235 (1964)
41. Meyerson, A.: Online facility location. In: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 426–431. IEEE Computer Society, Los Alamitos (2001)
42. Mirchandani, P.B., Francis, R.L.: Discrete Location Theory. Wiley, New York (1990)
43. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley, New York (1990)
44. Pál, M., Tardos, E., Wexler, T.: Facility location with nonuniform hard capacities. In: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 329–338. IEEE Computer Society, Los Alamitos (2001)
45. Qiu, L., Padmanabhan, V.N., Voelker, G.: On the placement of web server replicas. In: Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 1587–1596. IEEE Computer Society, Los Alamitos (2001)
46. Ravi, R., Sinha, A.: Hedging uncertainty: Approximation algorithms for stochastic optimization problems. Math. Program. **108**(1), 97–114 (2006)
47. Ravi, R., Sinha, A.: Integrated logistics: Approximation algorithms combining facility location and network design. In: Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization (IPCO). Lecture Notes in Computer Science, vol. 2337, pp. 212–229. Springer, Berlin (2002)
48. Ravi, R., Sinha, A.: Multicommodity facility location. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 342–349. SIAM, Philadelphia (2004)
49. Shmoys, D.B.: Approximation algorithms for facility location problems. In: Jansen, K., Khuller, S. (eds.) Approximation Algorithms for Combinatorial Optimization. Lecture Notes in Computer Science, vol. 1913, pp. 27–33. Springer, Berlin (2000)
50. Shmoys, D.B.: The design and analysis of approximation algorithms: Facility location as a case study. In: Thomas, R.R., Hosten, S., Lee, J. (eds) Proceedings of Symposia in Appl. Mathematics, vol. 61, pp. 85–97. AMS, Providence, RI, USA (2004)
51. Shmoys, D.B., Tardos, E., Aardal, K.: Approximation algorithms for facility location problems. In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC), pp. 265–274. ACM Press, New York (1997)
52. Svitkina, Z., Tardos, E.: Facility location with hierarchical facility costs. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm (SODA), pp. 153–161. SIAM, Philadelphia, PA, USA (2006)
53. Swamy, C., Kumar, A.: Primal-dual algorithms for connected facility location problems. Algorithmica **40**(4), 245–269 (2004)
54. Swamy, C., Shmoys, D.B.: Fault-tolerant facility location. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 735–736. SIAM, Philadelphia (2003)
55. Vygen, J.: Approximation algorithms for facility location problems (lecture notes). Technical report No. 05950-OR, Research Institute for Discrete Mathematics, University of Bonn (2005) http://www.or.uni-bonn.de/~vygen/fl.pdf
56. Zhang, J.: Approximating the two-level facility location problem via a quasi-greedy approach. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 808–817. SIAM, Philadelphia (2004). Also, Math. Program. **108**, 159–176 (2006)
57. Zhang, J., Chen, B., Ye, Y.: A multiexchange local search algorithm for the capacitated facility location problem. Math. Oper. Res. **30**(2), 389–403 (2005)

Failure Detectors

1996; Chandra, Toueg

RACHID GUERRAOUI
School of Computer and Communication Sciences, EPFL, Lausanne, Switzerland

Keywords and Synonyms

Partial synchrony; Time-outs; Failure information; Distributed oracles

Problem Definition

A distributed system is comprised of a collection of processes. The processes typically seek to achieve some common task by communicating through message passing or shared memory. Most interesting tasks require, at least at certain points of the computation, some form of *agreement* between the processes. An abstract form of such agreement is *consensus* where processes need to agree on a single value among a set of proposed values. Solving this seemingly elementary problem is at the heart of reliable

distributed computing and, in particular, of distributed database commitment, total ordering of messages, and emulations of many shared object types.

Fischer, Lynch, and Paterson's seminal result in the theory of distributed computing [13] says that consensus cannot be deterministically solved in an *asynchronous* distributed system that is prone to process failures. This impossibility holds consequently for all distributed computing problems which themselves rely on consensus.

Failures and *asynchrony* are fundamental ingredients in the consensus impossibility. The impossibility holds even if only *one* process *fails*, and it does so only by *crashing*, i. e., stopping its activities. Tolerating crashes is the least one would expect from a distributed system for the goal of distribution is in general to avoid single points of failures in centralized architectures. Usually, actual distributed applications exhibit more severe failures where processes could deviate arbitrarily from the protocol assigned to them.

Asynchrony refers to the absence of assumptions on process speeds and communication delays. This absence prevents any process from distinguishing a crashed process from a correct one and this inability is precisely what leads to the consensus impossibility. In practice, however, distributed systems are not completely asynchronous: some timing assumptions can typically be made. In the best case, if precise lower and upper bounds on communication delays and process speeds are assumed, then it is easy to show that consensus and related impossibilities can be circumvented despite the crash of any number of processes [20].

Intuitively, the way that such timing assumptions circumvent asynchronous impossibilities is by providing processes with *information about failures*, typically through *time-out* (or *heart-beat*) mechanisms, usually underlying actual distributed applications. Whereas certain information about failures can indeed be obtained in distributed systems, the accuracy of such information might vary from a system to another, depending on the underlying network, the load of the application, and the mechanisms used to detect failures. A crucial problem in this context is to characterize such information, in an abstract and precise way.

Key Results

The Failure Detector Abstraction

Chandra and Toueg [5] defined the *failure detector* abstraction as a simple way to capture failure information that is needed to circumvent asynchronous impossibilities, in particular the consensus impossibility. The model considered in [5] is a message passing one where processes can fail by *crashing*. Processes that crash stop their activities and do not recover. Processes that do not crash are said to be *correct*. At least one process is supposed to be correct in every execution of the system.

Roughly speaking, a failure detector is an oracle that provides processes with information about failures. The oracle is accessed in each computation step of a process and it provides the process with a value conveying some failure information. The value is picked from some set of values, called the *range* of the failure detector. For instance, the range could be the set of subsets of processes in the system, and each subset could depict the set of processes detected to have crashed, or considered to be correct. This would correspond to the situation where the failure detector is implemented using a time-out: every process q that does not communicate within some time period with some process p, would be included in subset of processes suspected of having crashed by p.

More specifically, a failure detector is a function, D, that associates to each *failure pattern*, F, a set of *failure detector histories* $\{H_i\} = D(F)$. Both the failure pattern and the failure detector history are themselves functions.

- A failure pattern F is a function that associates to each time t, the set of processes $F(t)$ that have indeed crashed by time t. This notion assumes the existence of a global clock, outside the control of the processes, as well as a specific concept of *crash* event associated with time. A set of failure pattern is called an *environment*.
- A failure detector history H is also a function, which associates to each process p and time t, some value v from the range of failure detector values. (The range of a failure detector D is denoted R_D.) This value v is said to be output by the failure detector D at process p and time t.

Two observations are in order.

- By construction, the output of a failure detector does not depend on the computation, i. e., on the actual steps performed by the processes, on their algorithm or the input of such algorithm. The output of the failure detector depends solely on the failure pattern, namely on whether and when processes crashed.
- A failure detector might associate several histories to each failure pattern. Each history represents a suite of possible combinations of outputs for the same given failure pattern. This captures the inherent nondeterminism of a failure detection mechanism. Such a mechanism is typically itself implemented as a distributed algorithm and the variations in communication delays for instance could lead the same mechanism

to output (even slightly) different information for the same failure pattern.

To illustrate these concepts, consider two classical examples of failure detectors.

1. The *perfect* failure detector outputs a subset of processes, i. e., the range of the failure detector is the set of subsets of processes in the system. When a process q is output at some time t at a process p, then q is said to be *detected* (of having crashed) by p. The *perfect* failure detector guarantees the two following properties:
   - Every process that crashes is eventually permanently detected;
   - No correct process is ever detected.
2. The *eventually strong* failure detector outputs a subset of processes: when a process q is output at some time t at a process p, then q is said to be *suspected* (of having crashed) by p. An *eventually strong* failure detector ensures the two following properties:
   - Every process that crashes is eventually suspected;
   - Eventually, some correct process is never suspected.

The *perfect* failure detector is *reliable*: if a process q is detected, then q has crashed. An *eventually strong* failure detector is *unreliable*: there never is any guarantee that the information that is output is accurate. The use of the the term *suspected* conveys that idea. The distinction between *unreliability* and *reliability* was precisely captured in [14] for the general context where the range of the failure detector can be arbitrary.

Consensus Algorithms

Two important results were established in [5].

Theorem 1 (Chandra-Toueg [5]) *There is an algorithm that solves consensus with a perfect failure detector.*

The theorem above implicitly says that if the distributed system provides means to implement perfect failure detection, then the consensus impossibility can be circumvented, even if all but one process crashes. In fact, the result holds for any failure pattern, i. e., in any environment.

The second theorem below relates the existence of a consensus algorithm to a resilience assumption. More specifically, the theorem holds in the *majority* environment, which is the set of failure patterns where more than half of the processes are correct.

Theorem 2 (Chandra-Toueg [5]) *There is an algorithm that implements consensus with an eventually strong failure detector in the majority environment.*

The algorithm underlying the result above is similar to *eventually synchronous* consensus algorithms [10] and share also some similarities with the *Paxos* algorithm [18].

It is shown in [5] that no algorithm using solely the *eventually strong* failure detector can solve consensus without the majority assumption. (This result is generalized to any unreliable failure detector in [14].) This resilience lower bound is intuitively due to the possibility of partitions in a message passing system where at least half of the processes can crash and failure detection is unreliable. In shared memory for example, no such possibility exists and consensus can be solved with the *eventually strong* failure [19].

Failure Detector Reductions

Failure detectors can be compared. A failure detector D_2 is said to be *weaker* than a failure detector D_1 if there is an asynchronous algorithm, called a *reduction* algorithm, which, using D_1, can emulate D_2. Three remarks are important here.

- The fact that the reduction algorithm is asynchronous means that it does not use any other source of failure information, besides D_1.
- Emulating failure detector D_2 means implementing a distributed variable that mimics the output that could be provided by D_2.
- The existence of a reduction algorithm depends on environment. Hence, strictly speaking, the fact that a failure detector is weaker than another one depends on the environment under consideration.

If failure detector D_1 is weaker than D_2, and vice et versa, then D_1 and D_2 are said to be *equivalent*. Else, if D_1 is weaker than D_2 and D_2 is not weaker than D_1, then D_1 is said to be *strictly weaker* than D_2. Again, strictly speaking, these notions depend on the considered environment.

The ability to compare failure detectors help define a notion of *weakest* failure detector to solve a problem. Basically, a failure detector D is the weakest to solve a problem P if the two following properties are satisfied:

- There is an algorithm that solves P using D.
- If there is an algorithm that solves P using some failure detector D', then D is weaker than D'.

Theorem 3 (Chandra-Hadzilacos-Toueg [4]) *The eventually strong failure detector is the weakest to solve consensus in the majority environment.*

The weakest failure detector to implement consensus in any environment was later established in [8].

Applications

A Practical Perspective

The identification of the failure detector concept had an impact on the design of reliable distributed architectures.

Basically, a failure detector can be viewed as a first class service of a distributed system, at the same level as a name service or a file service. Time-out and heartbeat mechanisms can thus be hidden under the failure detector abstraction, which can then export a unified interface to higher level applications, including consensus and state machine replication algorithms [2,11,21].

Maybe more importantly, a failure detector service can encapsulate synchrony assumptions: these can be changed without impact on the rest of the applications. Minimal synchrony assumptions to devise specific failure detectors could be explored leading to interesting theoretical results [1,7,12].

A Theoretical Perspective

A second application of the failure detector concept is a theory of distributed computability. Failure detectors enable to classify problems. A problem A is *harder* (resp. *strictly harder*) than problem B if the weakest failure detector to solve B is weaker (resp. strictly weaker) than the weakest failure detector to solve A. (This notion is of course parametrized by a specific environment.)

Maybe surprisingly, the induced failure detection reduction between problems does not exactly match the classical *black-box* reduction notion. For instance, it is well known that there is no asynchronous distributed algorithm that can use a *Queue* abstraction to implement a *Compare-Swap* abstraction in a system of $n > 2$ processes where $n - 1$ can fail by crashing [15]. In this sense, a *Compare-Swap* abstraction is strictly more powerful (in a *black-box* sense) than a *Queue* abstraction. It turns out that:

Theorem 4 (Delporte-Fauconnier-Guerraoui [9]) *The weakest failure detector to solve the Queue problem is also the weakest to solve the Compare-Swap problem in a system of $n > 2$ processes where $n - 1$ can fail by crashing.*

In a sense, this theorem indicates that reducibility as induced by the failure detector notion is different from the traditional *black-box* reduction.

Open Problems

Several issues underlying the failure detector notion are still open. One such issue consists in identifying the weakest failure detector to solve the seminal *set-agreement* problem [6]: a decision task where processes need to agree on up to k values, instead of a single value as in consensus. Three independent groups of researchers [3,16,22] proved the impossibility of solving this problem in an asynchronous system with k failures, generalizing the consensus impossibility [13]. Determining the weakest failure detector to circumvent this impossibility would clearly help understand the fundamentals of failure detection reducibility.

Another interesting research direction is to relate the complexity of distributed algorithm with the underlying failure detector [17]. Clearly, failure detectors circumvents asynchronous impossibilities, but to what extend do they boost the complexity of distributed algorithms? One would of course expect the complexity of a solution to a problem to be higher if the failure detector is weaker. But to what extend?

Cross References

- Asynchronous Consensus Impossibility
- Atomic Broadcast
- Causal Order, Logical Clocks, State Machine Replication
- Linearizability

Recommended Reading

1. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: On implementing Omega with weak reliability and synchrony assumptions. In: 22th ACM Symposium on Principles of Distributed Computing, pp. 306–314 (2003)
2. Bertier, M., Marin, O., Sens, P.: Performance analysis of a hierarchical failure detector. In: International Conference on Dependable Systems and Networks (DSN 2003), San Francisco, CA, USA, Proceedings, pp. 635–644. 22–25 June 2003
3. Boroswsky, E., Gafni E.: Generalized FLP impossibility result for t-resilient asynchronous computations. In: Proceedings of the 25th ACM Symposium on Theory of Computing, pp. 91–100, ACM Press
4. Chandra, T.D., Hadzilacos, V., Toueg, S.: The weakest failure detector for solving consensus. J. ACM **43**(4), 685–722 (1996)
5. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43**(2), 225–267 (1996)
6. Chauduri, S.: More choices allow more faults: Set consensus problems in totally asynchronous systems. Inf. Comput. **105**(1), 132–158 (1993)
7. Chen, W., Toueg, S., Aguilera, M.K.: On the quality of service of failure detectors. IEEE Trans. Comput. **51**(1), 13–32 (2002)
8. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R.: Failure detection lower bounds on registers and consensus. In: Proceedings of the 16th International Symposium on Distributed Computing, LNCS 2508 (2002)
9. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R.: Implementing atomic objects in a message passing system. Technical report, EPFL Lausanne (2005)
10. Dwork, C., Lynch, N.A., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
11. Felber, P., Guerraoui, R., Fayad, M.: Putting oo distributed programming to work. Commun. ACM **42**(11), 97–101 (1999)

12. Fernández, A., Jiménez, E., Raynal, M.: Eventual leader election with weak assumptions on initial knowledge, communication reliability and synchrony. In: Proc International Symposium on Dependable Systems and Networks (DSN), pp. 166–178 (2006)
13. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)
14. Guerraoui, R.: Indulgent algorithms. In: Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing, Portland, Oregon, USA, pp. 289–297, ACM, July 2000
15. Herlihy, M.: Wait-free synchronization. ACM Trans. Programm. Lang. Syst. **13**(1), 123–149 (1991)
16. Herlihy, M., Shavit, N.: The asynchronous computability theorem for *t*-resilient tasks. In: Proceedings of the 25th ACM Symposium on Theory of Computing, pp. 111–120, May 1993
17. Keidar, I., Rajsbaum, S.: On the cost of fault-tolerant consensus when there are no faults-a tutorial. In: Tutorial 21th ACM Symposium on Principles of Distributed Computing, July 2002
18. Lamport, L.: The Part-Time parliament. ACM Trans. Comput. Syst. **16**(2), 133–169 (1998)
19. Lo, W.-K., Hadzilacos, V.: Using failure detectors to solve consensus in asynchronous shared memory systems. In: Proceedings of the 8th International Workshop on Distributed Algorithms, LNCS 857, pp. 280–295, September 1994
20. Lynch, N.: Distributed Algorithms. Morgan Kauffman (1996)
21. Michel, R., Corentin, T.: In search of the holy grail: Looking for the weakest failure detector for wait-free set agreement. Technical Report TR 06-1811, INRIA, August 2006
22. Saks, M., Zaharoglou, F.: Wait-free *k*-set agreement is impossible: The topology of public knowledge. In: Proceedings of the 25th ACM Symposium on Theory of Computing, pp. 101–110, ACM Press, May 1993

False-Name-Proof Auction

2004; Yokoo, Sakurai, Matsubara

MAKOTO YOKOO
Information Science and Electrical Engineering,
Kyushu University,
Fukuoka, Japan

Keywords and Synonyms

False-name-proof auctions; Pseudonymous bidding; Robustness against false-name bids

Problem Definition

In Internet auctions, it is easy for a bidder to submit multiple bids under multiple identifiers (e. g., multiple e-mail addresses). If only one item/good is sold, a bidder cannot make any additional profit by using multiple bids. However, in combinatorial auctions, where multiple items/goods are sold simultaneously, submitting multiple bids under fictitious names can be profitable. A bid made under a fictitious name is called a *false-name bid*.

Here, use the same model as the GVA section. In addition, false-name bids are modeled as follows.

- Each bidder can use multiple identifiers.
- Each identifier is unique and cannot be impersonated.
- Nobody (except the owner) knows whether two identifiers belongs to the same bidder or not.

The goal is to design a *false-name-proof protocol*, i. e., a protocol in which using false-names is useless, thus bidders voluntarily refrain from using false-names.

The problems resulting from collusion have been discussed by many researchers. Compared with collusion, a false-name bid is easier to execute on the Internet since obtaining additional identifiers, such as another e-mail address, is cheap. False-name bids can be considered as a very restricted subclass of collusion.

Key Results

The Generalized Vickrey Auction (GVA) protocol is (dominant strategy) incentive compatible, i. e., for each bidder, truth-telling is a dominant strategy (a best strategy regardless of the action of other bidders) if there exists no false-name bids. However, when false-name bids are possible, truth-telling is no longer a dominant strategy, i. e., the GVA is not false-name-proof.

Here is an example, which is identical to Example 1 in the GVA section.

Example 1 Assume there are two goods a and b, and three bidders, bidder 1, 2, and 3, whose types are θ_1, θ_2, and θ_3, respectively. The evaluation value for a bundle $v(B, \theta_i)$ is determined as follows.

	$\{a\}$	$\{b\}$	$\{a, b\}$
θ_1	\$6	\$0	\$6
θ_2	\$0	\$0	\$8
θ_3	\$0	\$5	\$5

As shown in the GVA section, good a is allocated to bidder 1, and b is allocated to bidder 3. Bidder 1 pays \$3 and bidder 3 pays \$2.

Now consider another example.

Example 2 Assume there are only two bidders, bidder 1 and 2, whose types are θ_1 and θ_2, respectively. The evaluation value for a bundle $v(B, \theta_i)$ is determined as follows.

	$\{a\}$	$\{b\}$	$\{a, b\}$
θ_1	\$6	\$5	\$11
θ_2	\$0	\$0	\$8

In this case, the bidder 1 can obtains both goods, but he/she requires to pay \$8, since if bidder 1 does not participate, the social surplus would have been \$8. When bidder 1 does participate, bidder 1 takes everything and the social surplus except bidder 1 becomes 0. Thus, bidder 1 needs to pay the decreased amount of the social surplus, i. e., \$8.

However, bidder 1 can use another identifier, namely, bidder 3 and creates a situation identical to Example 1. Then, good a is allocated to bidder 1, and b is allocated to bidder 3. Bidder 1 pays \$3 and bidder 3 pays \$2. Since bidder 3 is a false-name of bidder 1, bidder 1 can obtain both goods by paying \$3 + \$2 = \$5. Thus, using a false-name is profitable for bidder 1.

The effects of false-name bids on combinatorial auctions are analyzed in [4]. The obtained results can be summarized as follows.

- As shown in the above example, the GVA protocol is not false-name-proof.
- There exists no false-name-proof combinatorial auction protocol that satisfies Pareto efficiency.
- If a surplus function of bidders satisfies a condition called *concavity*, then the GVA is guaranteed to be false-name-proof.

Also, a series of protocols that are false-name-proof in various settings have been developed: combinatorial auction protocols [2,3], multi-unit auction protocols [1], and double auction protocols [5].

Furthermore, in [2], a distinctive class of combinatorial auction protocols called a Price-oriented, Rationing-free (PORF) protocol is identified. The description of a PORF protocol can be used as a guideline for developing strategy/false-name proof protocols.

The outline of a PORF protocol is as follows:

1. For each bidder, the price of each bundle of goods is determined independently of his/her own declaration, while it depends on the declarations of other bidders. More specifically, the price of bundle (a set of goods) B for bidder i is determined by a function $p(B, \Theta_X)$, where Θ_X is a set of declared types by other bidders X.
2. Each bidder is allocated a bundle that maximizes his/her utility independently of the allocations of other bidders (i. e., rationing-free). The prices of bundles must be determined so that *allocation feasibility* is satisfied, i. e., no two bidders want the same item.

Although a PORF protocol appears to be quite different from traditional protocol descriptions, surprisingly, it is a sufficient and necessary condition for a protocol to be strategy-proof. Furthermore, if a PORF protocol satisfies the following additional condition, it is guaranteed to be false-name-proof.

Definition 1 (No Super-Additive price increase (NSA)) For any subset of bidders $S \subseteq N$ and $N' = N \setminus S$, and for $i \in S$, denote B_i as a bundle that maximizes i's utility, then $\sum_{i \in S} p(B_i, \bigcup_{j \in S \setminus \{i\}} \{\theta_j\} \cup \Theta_{N'}) \geq p(\bigcup_{i \in S} B_i, \Theta_{N'})$.

An intuitive description of this condition is that the price of buying a combination of bundles (the right side of the inequality) must be smaller than or equal to the sum of the prices for buying these bundles separately (the left side). This condition is also a necessary condition for a protocol to be false-name-proof, i. e., any false-name-proof protocol can be described as a PORF protocol that satisfies the NSA condition.

Here is a simple example of a PORF protocol that is false-name-proof. This protocol is called the Max Minimal-Bundle (M-MB) protocol [2]. To simplify the protocol description, a concept called a *minimal* bundle is introduced.

Definition 2 (minimal bundle) Bundle B is called minimal for bidder i, if for all $B' \subset B$ and $B' \neq B$, $v(B', \theta_i) < v(B, \theta_i)$ holds.

In this new protocol, the price of bundle B for bidder i is defined as follows:

- $p(B, \Theta_X) = \max_{B_j \subseteq M, j \in X} v(B_j, \theta_j)$, where $B \cap B_j \neq \emptyset$ and B_j is minimal for bidder j.

How this protocol works using Example 1 is described here. The prices for each bidder is determined as follows.

	$\{a\}$	$\{b\}$	$\{a, b\}$
bidder 1	\$8	\$8	\$8
bidder 2	\$6	\$5	\$6
bidder 3	\$8	\$8	\$8

The minimal bundle for bidder 1 is $\{a\}$, the minimal bundle for bidder 2 is $\{a, b\}$, and the minimal bundle for bidder 3 is $\{b\}$. The price of bundle $\{a\}$ for bidder 1 is equal to the largest evaluation value of conflicting bundles. In this case, the price is \$8, i. e., the evaluation value of bidder 2 for bundle $\{a, b\}$. Similarly, the price of bidder 2 for bundle $\{a, b\}$ is 6, i. e., the evaluation value of bidder 1 for bundle $\{a\}$. As a result, bundle $\{a, b\}$ is allocated to bidder 2.

It is clear that this protocol satisfies the allocation feasibility. For each good l, choose bidder j^* and bundle B_j^* that maximize $v(B_j, \theta_j)$ where $l \in B_j$ and B_j is minimal for bidder j. Then, only bidder j^* is willing to obtain a bundle that contains good l. For all other bidders, the price of a bundle that contains l is higher than (or equal to) his/her evaluation value.

Furthermore, it is clear that this protocol satisfies the NSA condition. In this pricing scheme, $p(B \cup B', \Theta_X) =$

$\max(p(B, \Theta_X), p(B', \Theta_X))$ holds for all B, B', and Θ_X. Therefore, the following formula holds

$$p\left(\bigcup_{i\in S} B_i, \Theta_X\right) = \max_{i\in S} p(B_i, \Theta_X) \leq \sum_{i\in S} p(B_i, \Theta_X)\,.$$

Furthermore, in this pricing scheme, prices increase monotonically by adding opponents, i. e., for all $X' \supseteq X$, $p(B, \Theta_{X'}) \geq p(B, \Theta_X)$ holds. Therefore, for each i, $p(B_i, \bigcup_{j\in S\setminus\{i\}}\{\theta_j\}\cup\Theta_{N'}) \geq p(B_i, \Theta_{N'})$ holds. Therefore, the NSA condition, i. e., $\sum_{i\in S} p(B_i, \bigcup_{j\in S\setminus\{i\}}\{\theta_j\}\cup\Theta_{N'}) \geq p(\bigcup_{i\in S} B_i, \Theta_{N'})$ holds.

Applications

In Internet auctions, using multiple identifiers (e. g., multiple e-mail addresses) is quite easy and identifying each participant on the Internet is virtually impossible. Combinatorial auctions have lately attracted considerable attention. When combinatorial auctions become widely used in Internet auctions, false-name-bids could be a serious problem.

Open Problems

It is shown that there exists no false-name-proof protocol that is Pareto efficient. Thus, it is inevitable to give up the efficiency to some extent. However, the theoretical lower-bound of the efficieny loss, i. e., the amount of the efficiency loss that is inevitabe for any false-name-proof protocol, is not identified yet. Also, the efficiency loss of existing false-name-proof protocols can be quite large. More efficient false-name-proof protocols in various settings are needed.

Cross References

► Generalized Vickrey Auction

Recommended Reading

1. Iwasaki, A., Yokoo, M., Terada, K.: A robust open ascending-price multi-unit auction protocol against false-name bids. Decis. Support. Syst. **39**, 23–39 (2005)
2. Yokoo, M.: The characterization of strategy/false-name proof combinatorial auction protocols: Price-oriented, rationing-free protocol. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence, pp. 733–739 (2003)
3. Yokoo, M., Sakurai, Y., Matsubara, S.: Robust combinatorial auction protocol against false-name bids. Artif. Intell. **130**, 167–181 (2001)
4. Yokoo, M., Sakurai, Y., Matsubara, S.: The effect of false-name bids in combinatorial auctions: New fraud in Internet auctions. Games Econ. Behav. **46**, 174–188 (2004)
5. Yokoo, M., Sakurai, Y., Matsubara, S.: Robust double auction protocol against false-name bids. Decis. Support. Syst. **39**, 23–39 (2005)

Fast Minimal Triangulation

2005; Heggernes, Telle, Villanger

Yngve Villanger
Department of Informatics,
University of Bergen,
Bergen, Norway

Keywords and Synonyms

Minimal fill problem

Problem Definition

Minimal triangulation is the addition of an inclusion minimal set of edges to an arbitrary undirected graph, such that a chordal graph is obtained. A graph is *chordal* if every cycle of length at least 4 contains an edge between two nonconsecutive vertices of the cycle.

More formally, Let $G = (V, E)$ be a simple and undirected graph, where $n = |V|$ and $m = |E|$. A graph $H = (V, E \cup F)$, where $E \cap F = \emptyset$ is a *triangulation* of G if H is chordal, and H is a *minimal* triangulation if there exists no $F' \subset F$, such that $H' = (V, E \cup F')$ is chordal. Edges in F are called *fill edges*, and a triangulation is minimal if and only if the removal of any single fill edge results in a chordless four cycle [10].

Since minimal triangulations were first described in the mid-1970s, a variety of algorithms have been published. A complete overview of these along with different characterizations of chordal graphs and minimal triangulations can be found in the survey of Heggernes et al. [5] on minimal triangulations. Minimal triangulation algorithms can roughly be partitioned into algorithms that obtain the triangulation through elimination orderings, and those that obtain it through vertex separators. Most of these algorithms have an $O(nm)$ running time, which becomes $O(n^3)$ for dense graphs. Among those that use elimination orderings, Kratsch and Spinrad's $O(n^{2.69})$-time algorithm [8] is currently the fastest one. The fastest algorithm is an $o(n^{2.376})$-time algorithm by Heggernes et al. [5]. This algorithm is based on vertex separators, and will be discussed further in the next section. Both the algorithm of Kratsch and Spinrad [8] and the algorithm of Heggernes et al. [5] use the matrix multiplication algorithm of Cop-

Algorithm FMT - Fast Minimal Triangulation
Input: An arbitrary graph $G = (V, E)$.
Output: A minimal triangulation G' of G.

Let Q_1, Q_2 and Q_3 be empty queues; Insert G into Q_1; $G' = G$;
repeat
  Construct a zero matrix M with a row for each vertex in V (columns are added later);
  while Q_1 is nonempty **do**
    Pop a graph $H = (U, D)$ from Q_1;
    Call **Algorithm Partition**(H) which returns a vertex subset $A \subset U$;
    Push vertex set A onto Q_3;
    for each connected component C of $H[U \setminus A]$ **do**
      Add a column in M such that $M(v, C) = 1$ for all vertices $v \in N_H(C)$;
      if there exists a non-edge uv in $H[N_H[C]]$ with $u \in C$ **then**
        Push $H_C = (N_H[C], D_C)$ onto Q_2, where $uv \notin D_C$ if $u \in C$ and $uv \notin D$;
  Compute MM^T;
  Add to G' the edges indicated by the nonzero elements of MM^T;
  while Q_3 is nonempty **do**
    Pop a vertex set A from Q_3;
    if $G'[A]$ is not complete **then** Push $G'[A]$ onto Q_2;
  Swap names of Q_1 and Q_2;
until Q_1 is empty

Fast Minimal Triangulation, Figure 1
Fast minimal triangulation algorithm

persmith and Winograd [3] to obtain an $o(n^3)$-time algorithm.

Key Results

For a vertex set $A \subset V$, the subgraph of G induced by A is $G[A] = (A, W)$, where $uv \in W$ if $u, v \in A$ and $uv \in E\}$). The closed neighborhood of A is $N[A] = U$, where $u, v \in U$ for every $uv \in E$, where $u \in A\}$ and $N(A) = N[A] \setminus A$. A is called a *clique* if $G[A]$ is a complete graph. A vertex set $S \subset V$ is called a *separator* if $G[V \setminus S]$ is disconnected, and S is called a *minimal* separator if there exists a pair of vertices $a, b \in V \setminus S$ such that a, b are contained in different connected components of $G[V \setminus S]$, and in the same connected component of $G[V \setminus S']$ for any $S' \subset S$. A vertex set $\Omega \subseteq V$ is a *potential maximal clique* if there exists no connected component of $G[V \setminus \Omega]$ that contains Ω in its neighborhood, and for every vertex pair $u, v \in \Omega$, uv is an edge or there exists a connected component of $G[V \setminus \Omega]$ that contains both u and v in its neighborhood.

From the results in [1,7], the following recursive minimal triangulation algorithm is obtained. Find a vertex set A which is either a minimal separator or a potential maximal clique. Complete $G[A]$ into a clique. Recursively for each connected component C of $G[V \setminus A]$ where $G[N[C]]$ is not a clique, find a minimal triangulation of $G[N[C]]$. An important property here is that the set of connected components of $G[V \setminus A]$ defines independent minimal triangulation problems.

The recursive algorithm just described defines a tree, where the given input graph G is the root node, and where each connected component of $G[V \setminus A]$ becomes a child of the root node defined by G. Now continue recursively for each of the subproblems defined by these connected components. A node H which is actually a subproblem of the algorithm is defined to be at *level i*, if the distance from H to the root in the tree is i. Notice that all subproblems at the same level can be triangulated independently. Let k be the number of levels. If this recursive algorithm can be completed for every subgraph at each level in $O(f(n))$ time, then this trivially provides an $O(f(n) \cdot k)$-time algorithm.

The algorithm in Fig. 1 uses queues to obtain this level-by-level approach, and matrix multiplication to complete all the vertex separators at a given level in $O(n^\alpha)$ time, where $\alpha < 2.376$ [3]. In contrast to the previously de-

Algorithm Partition
Input: A graph $H = (U, D)$ (a subproblem popped from Q_1).
Output: A subset A of U such that either $A = N[K]$ for some connected $H[K]$ or A is a potential maximal clique of H (and G').

Part I: defining *P*
Unmark all vertices of H; $k = 1$;
while there exists an unmarked vertex u **do**
  if $\mathcal{E}_{\bar{H}}(U \setminus N_H[u]) < \frac{2}{5}|\bar{E}(H)|$ **then** Mark u as an **s**-vertex (stop vertex);
  else
    $C_k = \{u\}$; Mark u as a **c**-vertex (component vertex);
    while there exists a vertex $v \in N_H[C_k]$ which is unmarked or marked as an **s**-vertex **do**
      if $\mathcal{E}_{\bar{H}}(U \setminus N_H[C_k \cup \{v\}]) \geq \frac{2}{5}|\bar{E}(H)|$ **then**
        $C_k = C_k \cup \{v\}$; Mark v as a **c**-vertex (component vertex);
      else
        Mark v as a **p**-vertex (potential maximal clique vertex); Associate v with C_k;
    $k = k + 1$;
P = the set of all **p**-vertices and **s**-vertices;

Part II: defining *A*
if $H[U \setminus P]$ has a full component C **then** $A = N_H[C]$;
else if there exist two non-adjacent vertices u, v such that u is an **s**-vertex and v is an **s**-vertex or a **p**-vertex **then** $A = N_H[u]$;
else if there exist two non-adjacent **p**-vertices u and v, where u is associated with C_i and v is associated with C_j and $u \notin N_H(C_j)$ and $v \notin N_H(C_i)$ **then** $A = N_H[C_i \cup \{u\}]$;
else $A = P$;

Fast Minimal Triangulation, Figure 2
Partitioning algorithm. Let $\bar{E}(H) = W$, where $uv \in W$ if $uv \notin D$ be the set of nonedges of H. Define $\mathcal{E}_{\bar{H}}(S)$ to be the sum of degrees in $\bar{H} = (U, \bar{E})$ of vertices in $S \subseteq U = V(H)$

scribed recursive algorithm, the algorithm in Fig. 1 uses a partitioning subroutine that either returns a *set* of minimal separators or a potential maximal clique.

Even though all subproblems at the same level can be solved independently they may share vertices and edges, but no nonedges (i. e., pair of vertices that are not adjacent). Since triangulation involves edge addition, the number of nonedges will decrease for each level, and the sum of nonedges for all subproblems at the same level will never exceed n^2. The partitioning algorithm in Fig. 2 exploits this fact and has an $O(n^2 - m)$ running time, which sums up to $O(n^2)$ for each level. Thus, each level in the fast minimal triangulation algorithm given in Fig. 1 can be completed in $O(n^2 + n^\alpha)$ time, where $O(n^\alpha)$ is the time needed to compute MM^T. The partitioning algorithm in Fig. 2 actually finds a set A that defines a set of minimal separators, such that no subproblem contains more than four fifths of the nonedges in the input graph. As a result, the number of levels in the fast minimal triangulation algorithm is at most $\log_{4/5}(n^2) = 2\log_{4/5}(n)$, and the running time $O(n^\alpha \log n)$ is obtained.

Applications

The first minimal triangulation algorithms were motivated by the need to find good pivotal orderings for Gaussian elimination. Finding an optimal ordering is equivalent to solving the minimum triangulation problem, which is a nondeterministic polynomial-time hard problem. Since any minimum triangulation is also a minimal triangulation, and minimal triangulations can be found in polynomial time, then the set of minimal triangulations can be a good place to search for a pivotal ordering.

Probably because of the desired goal, the first minimal triangulation algorithms were based on orderings, and produced an ordering called a minimal elimination ordering. The problem of computing a minimal triangulation has received increasing attention since then, and several

new applications and characterizations related to the vertex separator properties have been published. Two of the new applications are computing the tree-width of a graph, and reconstructing evolutionary history through phylogenetic trees [6]. The new separator-based characterizations of minimal triangulations have increased the knowledge of minimal triangulations [1,7,9]. One result based on these characterizations is an algorithm that computes the tree-width of a graph in polynomial time if the number of minimal separators is polynomially bounded [2]. A second application is faster exact (exponential-time) algorithms for computing the tree-width of a graph [4].

Open Problems

The algorithm described shows that a minimal triangulation can be found in $O((n^2 + n^\alpha) \log\ n)$ time, where $O(n^\alpha)$ is the time required to preform an $n \times n$ binary matrix multiplication. As a result, any improved binary matrix multiplication algorithm will result in a faster algorithm for computing a minimal triangulation. An interesting question is whether or not this relation goes the other way as well. Does there exist an $O((n^2 + n^\beta) f(n))$ algorithm for binary matrix multiplication, where $O(n^\beta)$ is the time required to find a minimal triangulation and $f(n) = o(n^{\alpha-2})$ or at least $f(n) = O(n)$. A possibly simpler and related question previously asked in [8] is: Is it at least as hard to compute a minimal triangulation as to determine whether a graph contains at least one triangle? A more algorithmic question is if there exists an $O(n^2 + n^\alpha)$-time algorithm for computing a minimal triangulation.

Cross References

► Treewidth of Graphs

Recommended Reading

1. Bouchitté, V., Todinca, I.: Treewidth and minimum fill-in: Grouping the minimal separators. SIAM J. Comput. **31**, 212–232 (2001)
2. Bouchitté, V., Todinca, I.: Listing all potential maximal cliques of a graph. Theor. Comput. Sci. **276**(1–2), 17–32 (2002)
3. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symb. Comput. **9**(3), 251–280 (1990)
4. Fomin, F.V., Kratsch, D., Todinca, I.: Exact (exponential) algorithms for treewidth and minimum fill-in. In: ICALP of LNCS, vol. 3142, pp. 568–580. Springer, Berlin (2004)
5. Heggernes, P., Telle, J.A., Villanger, Y.: Computing minimal triangulations in time $O(n^\alpha \log n) = o(n^{2.376})$. SIAM J. Discret. Math. **19**(4), 900–913 (2005)
6. Huson, D.H., Nettles, S., Warnow, T.: Obtaining highly accurate topology estimates of evolutionary trees from very short sequences. In: RECOMB, 1999, pp. 198–207
7. Kloks, T., Kratsch, D., Spinrad, J.: On treewidth and minimum fill-in of asteroidal triple-free graphs. Theor. Comput. Sci. **175**, 309–335 (1997)
8. Kratsch, D., Spinrad, J.: Minimal fill in $O(n^{2.69})$ time. Discret. Math. **306**(3), 366–371 (2006)
9. Parra, A., Scheffler, P.: Characterizations and algorithmic applications of chordal graph embeddings. Discret. Appl. Math. **79**, 171–188 (1997)
10. Rose, D., Tarjan, R.E., Lueker, G.: Algorithmic aspects of vertex elimination on graphs. SIAM J. Comput. **5**, 146–160 (1976)

Fault-Tolerant Quantum Computation

1996; Shor, Aharonov, Ben-Or, Kitaev

Ben W. Reichardt
Department of Computer Science,
University of California, Berkeley, CA, USA

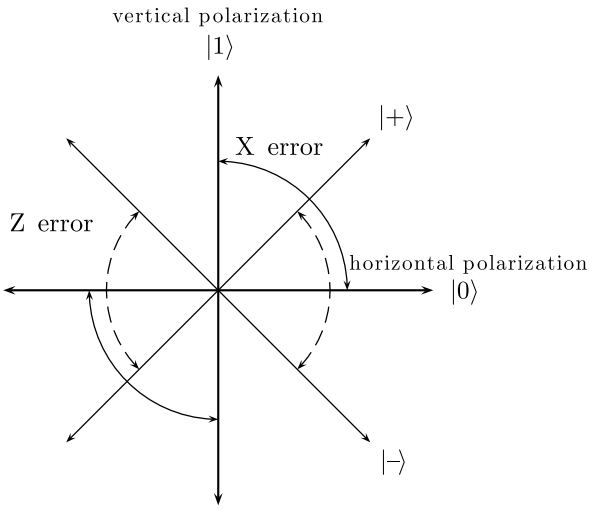
Keywords and Synonyms

Quantum noise threshold

Problem Definition

Fault tolerance is the study of reliable computation using unreliable components. With a given noise model, can one still reliably compute? For example, one can run many copies of a classical calculation in parallel, periodically using majority gates to catch and correct faults. Von Neumann showed in 1956 that if each gate fails independently with probability p, flipping its output bit $0 \leftrightarrow 1$, then such a fault-tolerance scheme still allows for arbitrarily reliable computation provided p is below some constant threshold (whose value depends on the model details) [10].

In a quantum computer, the basic gates are much more vulnerable to noise than classical transistors – after all, depending on the implementation, they are manipulating single electron spins, photon polarizations and similarly fragile subatomic particles. It might not be possible to engineer systems with noise rates less than 10^{-2}, or perhaps 10^{-3}, per gate. Additionally, the phenomenon of entanglement makes quantum systems *inherently* fragile. For example, in Schrödinger's cat state – an equal superposition between a living cat and a dead cat, often idealized as $1/\sqrt{2}(|0^n\rangle + |1^n\rangle)$ – an interaction with just one quantum bit ("qubit") can collapse, or decohere, the entire system. Fault-tolerance techniques will therefore be essential for achieving the considerable potential of quantum computers. Practical fault-tolerance techniques will need to control high noise rates and do so with low overhead, since qubits are expensive.



Fault-Tolerant Quantum Computation, Figure 1
Bit-flip X errors flip 0 and 1. In a qubit, $|0\rangle$ and $|1\rangle$ might be represented by horizontal and vertical polarization of a photon, respectively. Phase-flip Z errors flip the $\pm 45°$ polarized states $|+\rangle$ and $|-\rangle$

Quantum systems are continuous, not discrete, so there are many possible noise models. However, the essential features of quantum noise for fault-tolerance results can be captured by a simple discrete model similar to the one Von Neumann used. The main difference is that, in addition to bit-flip X errors which swap 0 and 1, there can also be phase-flip Z errors which swap $|+\rangle \equiv 1/\sqrt{2}(|0\rangle + |1\rangle)$ and $|-\rangle \equiv 1/\sqrt{2}(|0\rangle - |1\rangle)$ (Fig. 1). A noisy gate is modeled as a perfect gate followed by independent introduction of X, Z, or Y (which is both X and Z) errors with respective probabilities p_X, p_Z, p_Y. One popular model is independent depolarizing noise ($p_X = p_Z = p_Y \equiv p/3$); a depolarized qubit is completely randomized.

Faulty measurements and preparations of single-qubit states must additionally be modeled, and there can be memory noise on resting qubits. It is often assumed that measurement results can be fed into a classical computer that works perfectly and dynamically adjusts the quantum gates, although such control is not necessary. Another common, though unnecessary, assumption is that any pair of qubits in the computer can interact; this is called a *nonlocal* gate. In many proposed quantum computer implementations, however, qubit mobility is limited so gates can be applied only locally, between physically nearby qubits.

Key Results

The key result in fault tolerance is the existence of a noise *threshold*, for certain noise and computational models. The noise threshold is a positive, constant noise rate (or set of model parameters) such that with noise below this rate, reliable computation is possible. That is, given an inputless quantum circuit C of perfect gates, there exists a "simulating" circuit FTC of faulty gates such that with probability at least 2/3, say, the measured output of C agrees with that of FTC. Moreover, FTC should be only polynomially larger than C.

A quantum circuit with N gates can a priori tolerate only $O(1/N)$ error per gate, since a single failure might randomize the entire output. In 1996, Shor showed how to tolerate $O(1/\text{poly}(\log N))$ error per gate by encoding each qubit into a poly$(\log N)$-sized quantum error-correcting code; and then implementing each gate of the desired circuit directly on the encoded qubits, alternating computation and error-correction steps (similar to Von Neumann's scheme) [8]. Shor's result has two main technical pieces:

1. The discovery of quantum error-correcting codes (QECCs) was a major result. Remarkably, even though quantum errors can be continuous, codes that correct discrete errors suffice. (Measuring the syndrome of a code block projects into a discrete error event.) The first quantum code, discovered by Shor, was a nine-qubit code consisting of the concatenation of the three-qubit repetition code $|0\rangle \mapsto |000\rangle, |1\rangle \mapsto |111\rangle$ to protect against bit-flip errors, with its dual $|+\rangle \mapsto |++\rangle, |-\rangle \mapsto |---\rangle$ to protect against phase-flip errors. Since then, many other QECCs have been discovered. Codes like the nine-qubit code that can correct bit- and phase-flip errors separately are known as Calderbank-Shor-Steane (CSS) codes, and have quantum codewords which are simultaneously superpositions over codewords of classical codes in both the $|0/1\rangle$ and $|+/-\rangle$ bases.
2. QECCs allow for quantum memory or for communicating over a noisy channel. For computation, however, it must be possible to compute on encoded states without first decoding. An operation is said to be *fault tolerant* if it cannot cause correlated errors within a code block. With the n-bit majority code, all classical gates can be applied *transversely* – an encoded gate can be implemented by applying the unencoded gate to bits i of each code block, $1 \le i \le n$. This is fault tolerant because a single failure affects at most one bit in each block, thus failures can't spread too quickly. For CSS quantum codes, the controlled-NOT gate CNOT: $|a, b\rangle \mapsto |a, a \oplus b\rangle$ can similarly be applied transversely. However, the CNOT gate by itself is not universal, so Shor also gave a fault-tolerant implementation of the Toffoli gate $|a, b, c\rangle \mapsto |a, b, c \oplus (a \wedge b)\rangle$.

Procedures are additionally needed for error correction using faulty gates, and for the initial preparation step. The encoding of $|0\rangle$ will be a highly entangled state and difficult to prepare (unlike 0^n for the classical majority code).

However, Shor did not prove the existence of a *constant* tolerable noise rate, a noise threshold. Several groups – Aharonov/Ben-Or, Kitaev, and Knill/Laflamme/Zurek – each had the idea of using smaller codes, and *concatenating* the procedure repeatedly on top of itself. Intuitively, with a distance-three code (i. e., code that corrects any one error), one expects the "effective" logical error rate of an encoded gate to be at most cp^2 for some constant c, because one error can be corrected but two errors cannot. The effective error rate for a twice-encoded gate should then be at most $c(cp^2)^2$; and since the effective error rate is dropping doubly-exponentially fast in the number of levels of concatenation, the overhead in achieving a $1/N$ error rate is only poly(log N). The threshold for improvement, $cp^2 < p$, is $p < 1/c$. However, this rough argument is not rigorous, because the effective error rate is ill defined, and logical errors need not fit the same model as physical errors (for example, they will not be independent).

Aharonov and Ben-Or, and Kitaev gave independent rigorous proofs of the existence of a positive constant noise threshold, in 1997 [1,5].

Broadly, there has since been progress on two fronts of the fault-tolerance problem:

1. First, work has proceeded on extending the set of noise and computation models in which a fault-tolerance threshold is known to exist. For example, correlated or even adversarial noise, leakage errors (where a qubit leaves the $|0\rangle, |1\rangle$ subspace), and non-Markovian noise (in which the environment has a memory) have all been shown to be tolerable in theory, even with only local gates.
2. Threshold existence proofs establish that building a working quantum computer is possible *in principle*. Physicists need only engineer quantum systems with a low enough constant noise rate. But realizing the potential of a quantum computer will require *practical* fault-tolerance schemes. Schemes will have to tolerate a high noise rate (not just some constant) and do so with low overhead (not just polylogarithmic). However, rough estimates of the noise rate tolerated by the original existence proofs are not promising – below 10^{-6} noise per gate. If the true threshold is only 10^{-6}, then building a quantum computer will be next to impossible. Therefore, second, there has been substantial work on optimizing fault-tolerance schemes primarily in order to improve the tolerable noise rate. These optimizations are typically evaluated with simulations and heuristic analytical models. Recently, though, Aliferis, Gottesman and Preskill have developed a method to prove reasonably good threshold lower bounds, up to 2×10^{-4}, based on counting "malignant" sets of error locations [3].

In a breakthrough, Knill has constructed a novel fault-tolerance scheme based on very efficient distance-*two* codes [6]. His codes cannot correct any errors and the scheme uses extensive postselection on no detected errors – i. e., on detecting an error, the enclosing subroutine is restarted. He has estimated a threshold above 3% per gate, an order of magnitude higher than previous estimates. Reichardt has proved a threshold lower bound of 10^{-3} for a similar scheme [7], somewhat supporting Knill's high estimate. However, reliance on postselection leads to an enormous overhead at high error rates, greatly limiting practicality. (A classical fault-tolerance scheme based on error detection could not be efficient, but quantum teleportation allows Knill's scheme to be at least theoretically efficient.) There seems to be tradeoff between the tolerable noise rate and the overhead required to achieve it.

There are several complementary approaches to quantum fault tolerance. For maximum efficiency, it is wise to exploit any known noise structure before switching to general fault-tolerance procedures. Specialized techniques include careful quantum engineering, techniques from nuclear magnetic resonance (NMR) such as dynamical decoupling and composite pulse sequences, and decoherence-free subspaces. For very small quantum computers, such techniques may give sufficient noise protection.

It is possible that an inherently reliable quantum-computing device will be engineered or discovered, like the transistor for classical computing, and this is the goal of *topological* quantum computing [4].

Applications

As quantum systems are noisy and entanglement-fragile, fault-tolerance techniques will probably be essential in implementing any quantum algorithms – including, e. g., efficient factoring and quantum simulation.

The quantum error-correcting codes originally developed for fault-tolerance have many other applications, including for example quantum key distribution.

Open Problems

Dealing with noise may turn out to be the most daunting task in building a quantum computer. Currently, physicists' low-end estimates of achievable noise rates are

only slightly below theorists' high-end (mostly simulation-based) estimates of tolerable noise rates, at reasonable levels of overhead. However these estimates are made with different noise models – most simulations are based on the simple independent depolarizing noise model, and threshold lower bounds for more general noise are much lower. Also, both communities may be being too optimistic. Unanticipated noise sources may well appear as experiments progress. The probabilistic noise models used by theorists in simulations may not match reality closely enough, or the overhead/threshold tradeoff may be impractical. It is not clear if fault-tolerant quantum computing will work in practice, unless inefficiencies are wrung out of the system. Developing more efficient fault-tolerance techniques is a major open problem. Quantum system engineering, with more realistic simulations, will be required to understand better various tradeoffs and strategies for working with gate locality restrictions.

The gaps between threshold upper bounds, threshold estimates and rigorously proven threshold lower bounds are closing, at least for simple noise models. Our understanding of what to expect with more realistic noise models is less developed, though. One current line of research is in extending threshold proofs to more realistic noise models – e. g., [2]. A major open question here is whether a noise threshold can be shown to even *exist* where the bath Hamiltonian is unbounded – e. g., where system qubits are coupled to a non-Markovian, harmonic oscillator bath. Even when a threshold is known to exist, rigorous threshold lower *bounds* in more general noise models may still be far too conservative (according to arguments, mostly intuitive, known as "twirling") and, since simulations of general noise models are impractical, new ideas are needed for more efficient analyzes.

Theoretically, it is of interest what is the best asymptotic overhead in the simulating circuit FTC versus C? Overhead can be measured in terms of size N and depth/time T. With concatenated coding, the size and depth of FTC are $O(N\text{poly}\log N)$ and $O(T\text{poly}\log N)$, respectively. For classical circuits C, however, the depth can be only $O(T)$. It is not known if the quantum depth overhead can be improved.

Experimental Results

Fault-tolerance schemes have been simulated for large quantum systems, in order to obtain threshold estimates. For example, extensive simulations including geometric locality constraints have been run by Thaker et al. [9].

Error correction using very small codes has been experimentally verified in the lab.

URL to Code

Andrew Cross has written and distributes code for giving Monte Carlo estimates of and rigorous lower bounds on fault-tolerance thresholds: http://web.mit.edu/awcross/www/qasm-tools/. Emanuel Knill has released *Mathematica* code for estimating fault-tolerance thresholds for certain postselection-based schemes: http://arxiv.org/e-print/quant-ph/0404104.

Cross References

► Quantum Error Correction

Recommended Readings

1. Aharonov, D., Ben-Or, M.: Fault-tolerant quantum computation with constant error rate. In: Proc. 29th ACM Symp. on Theory of Computing (STOC), pp. 176–188, (1997). quant-ph/9906129
2. Aharonov, D., Kitaev, A.Y., Preskill, J.: Fault-tolerant quantum computation with long-range correlated noise. Phys. Rev. Lett. **96**, 050504 (2006). quant-ph/0510231
3. Aliferis, P., Gottesman, D., Preskill, J.: Quantum accuracy threshold for concatenated distance-3 codes. Quant. Inf. Comput. **6**, 97–165 (2006). quant-ph/0504218
4. Freedman, M.H., Kitaev, A.Y., Larsen, M.J., Wang, Z.: Topological quantum computation. Bull. AMS **40**(1), 31–38 (2002)
5. Kitaev, A.Y.: Quantum computations: algorithms and error correction. Russ. Math. Surv. **52**, 1191–1249 (1997)
6. Knill, E.: Quantum computing with realistically noisy devices. Nature **434**, 39–44 (2005)
7. Reichardt, B.W.: Error-detection-based quantum fault tolerance against discrete Pauli noise. Ph. D. thesis, University of California, Berkeley (2006). quant-ph/0612004
8. Shor, P.W.: Fault-tolerant quantum computation. In: Proc. 37th Symp. on Foundations of Computer Science (FOCS) (1996). quant-ph/9605011
9. Thaker, D.D., Metodi, T.S., Cross, A.W., Chuang, I.L., Chong, F.T.: Quantum memory hierarchies: Efficient designs to match available parallelism in quantum computing. In: Proc. 33rd. Int. Symp. on Computer Architecture (ISCA), pp. 378–390 (2006) quant-ph/0604070
10. von Neumann, J.: Probabilistic logic and the synthesis of reliable organisms from unreliable components. In: Shannon, C.E., McCarthy, J. (eds.) Automata Studies, pp. 43–98. Princeton University Press, Princeton (1956)

File Caching and Sharing

► Data Migration
► Online Paging and Caching
► P2P

Floorplan and Placement

1994; Kajitani, Nakatake, Murata, Fujiyoshi

YOJI KAJITANI
Department of Information and Media Sciences,
The University of Kitakyushu, Kitakyushu, Japan

Keywords and Synonyms

Layout; Alignment; Packing; Dissection

Problem Definition

The problem is concerned with efficient coding of the constraint that defines the placement of objects on a plane without mutual overlapping. This has numerous motivations, especially in the design automation of integrated semiconductor chips, where almost hundreds of millions of rectangular modules shall be placed within a small rectangular area (chip). Until 1994, the only known coding efficient in computer aided design was *Polish-Expression* [1]. However, this can only handle a limited class of placements of the *slicing structure*. In 1994 Nakatake, Fujiyoshi, Murata, and Kajitani [2], and Murata, Fujiyoshi, Nakatake, and Kajitani [3] were finally successful to answer this long-standing problem in two contrasting ways. Their code names are *Bounded-Sliceline-Grid* (BSG) for floorplanning and *Sequence-Pair* (SP) for placement.

Notations

1. *Floorplanning, placement, compaction, packing, layout:* Often they are used as exchangeable terms. However, they have their own implications to be used in the following context. *Floorplanning* concerns the design of the plane by restricting and partitioning a given area on which objects are able to be properly *placed*. *Packing* tries a placement with an intention to reduce the area occupied by the objects. *Compaction* supports packing by pushing objects to the center of the placement. The result, including other environments, is the *layout*. BSG and SP are paired concepts, the former for "floorplanning", the latter for "placement".

2. *ABLR-relation:* The objects to be placed are assumed rectangles in this entry though they could be more general depending on the problem. For two objects p and q, p is said to be *above* q (denoted as pAq) if the bottom edge (boundary) of p is above the top edge of q. Other relations with respect to "*below*" (pBq), "*left-of*" (pLq), and "*right-of*" (pRq) are analogously defined. These four relations are generally called *ABLR-relations*. A placement without mutual overlapping of objects is said to be *feasible*. Trivially, a placement is feasible if and only if every pair of objects is in one of ABLR-relations. The example in Fig. 1 will help these definitions.

It must be noted that a pair of objects may satisfy two ABLR-relations simultaneously, but not three. Furthermore, an arbitrary set of ABLR-relations is not necessarily *consistent* for any feasible placement. For example, any set of ABLR-relations including relations (pAq), (qAr), and (rAp) is not consistent.

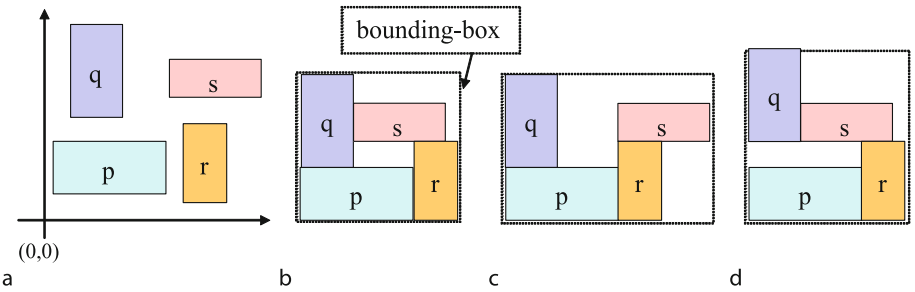
3. *Compaction:* Given a placement, its *bounding-box* is the minimum rectangle that encloses all the objects. A placement of objects is evaluated by the smallness of the bounding box's area, abbreviated as the *bb-area*. An ABLR-relation set is also evaluated by the minimum bb-area of all the placements that satisfy the set. However, given a consistent ABLR-relation set, the corresponding placement is not unique in general. Still, the minimum bb-area is easily obtained by a common technique called the "Longest-Path Algorithm". (See for example [4].)

Consider the placement whose objects are all inside the 1st quadrant of the *xy*-coordinate system, without loss of generality with respect to minimizing the bb-area. It is evident that if a given ABLR-relation set is feasible, there is an object that has no object left or below it. Place it such that its left-bottom corner is at the origin. From the remaining objects, take one that has no object left of or below it. Place it as leftward and downward as long as any ABLR-relation with already fixed objects is not violated. See Fig. 1 to catch the concept, where the ABLR-relation set is the one obtained the placement in (a) (so that it is trivially feasible). It is possible to obtain different ABLR-relation sets, according to which compaction would produce different placements.

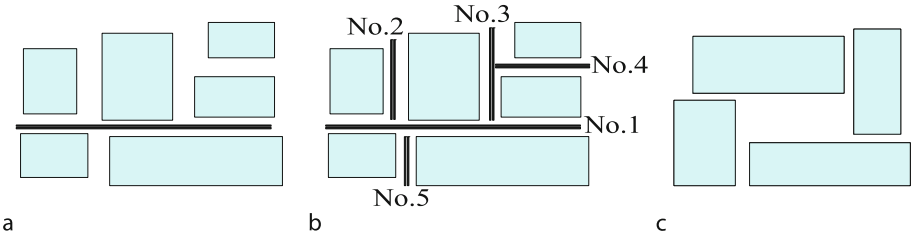
4. *Slice-line:* If it is possible to draw a straight horizontal line or vertical line to separate the objects into two groups, the line is said a *slice-line*. If each group again has a slice-line, and so does recursively, the placement is said to be a *slicing structure*. Figure 2 shows placements of slicing and non-slicing structures.

5. *Spiral:* Two structures each consisting of four line segments connected by a *T-junction* as shown in Fig. 3a are *spirals*. Their regular alignment in the 1st quadrant as shown in (b) is the *Bounded-Sliceline-Grid* or BSG. A BSG is a *floorplan*, or a *T-junction dissection*, of the rectangular area into rectangular regions called *rooms*. It is denoted as an $n \times m$ BSG if the numbers of rows and columns of its rooms are n and m, respectively. According to the left-bottom room being p-type or q-type, the BSG is said to be p-type or q-type, respectively.

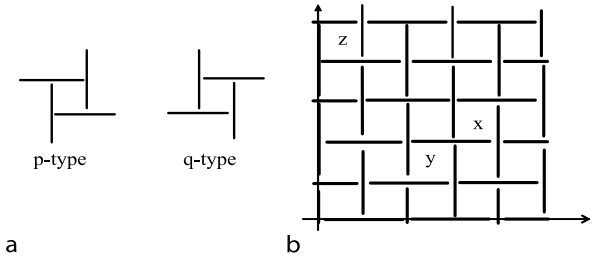
In a BSG, take two rooms x and y. The ABLR-relations between them are all that is defined by the rule: If the bottom segment of x is the top segment of y (Fig. 3), room x



Floorplan and Placement, Figure 1
a A feasible placement whose ABLR-relations could be observed differently. b Compacted placement if ABLR-relations are (qLr), (sAp), ... Its Sequence-Pair is SP = (qspr,pqrs) and Single-Sequence is SS = (2413). c Compacted placement for (qLr), (sRp), ... SP = (qpsr,pqrs). SS = (2143). d Compacted placement if (qAr), (sAp), ... SP = (qspr,prqs). SS = (3412)



Floorplan and Placement, Figure 2
a A placement with a slice-line. b A slicing structure since a slice-line can be found in each *i*th hierachy No. *k*(*k* = 1, 2, 3, 4). c A placement that has no slice-line



Floorplan and Placement, Figure 3
a Two types of the spiral structure (2) 5 × 5*p*-type Bounded-Sliceline-Grid (BSG)

is above room y. Furthermore, *Transitive-Law* is assumed: If "x is above y" and "z is above x", then "z is above y".

Other relations are analogously defined.

Lemma 1 *A room is in a unique ABLR-relation with every other room.*

An $n \times n$ BSG has n^2 rooms. A BSG-assignment is a *one-to-one mapping* of n objects into the rooms of $n \times n$ BSG. ($n^2 - n$ rooms remain vacant.)

After a BSG-assignment, a pair of two objects inherits the same ABLR-relation as the ABLR-relation defined between corresponding rooms. In Fig. 3, if x, y, and z are the names of objects, are ABLR-relations among them as {(xAy), (xRz), (yBx), (yBz), (zLx), (zAy)}.

Key Results

The input is n objects that are rectangles of arbitrary sizes. The main concern is the *solution space*, the collection of distinct consistent ABLR-relation sets, to be generated by BSG or SP.

Theorem 2 ([4,5])

1) *For any feasible ABLR-relation set, there is a BSG-assignment into $n \times n$ BSG of any type that generates the same ABLR-relation set.*
2) *The size $n \times n$ is a minimum: if the number of rows or columns is less than n, there is a feasible ABLR-relation set that is not obtained by any BSG-assignment.*

The proof to 1) is not trivial [5](Appendix). The number of solutions is ${}_{n^2}C_n$. A remarkable feature of an $n \times n$ BSG is that any ABLR-relation set of n objects is generated by a proper BSG-assignment. By this property, BSG is said to be *universal* [11].

In contrast to the BSG-based generation of consistent ABLR-relation sets, SP directly imposes the ABLR-relations on objects.

A pair of permutations of object names, represented as (Γ^+, Γ^-), is called the Sequence-Pair, or SP. See Fig. 1. An SP is decoded to a unique ABLR-relation set by the rule:

Consider a pair (x, y) of names such that x is before y in Γ^-. Then (xLy) or (xAy) if x is before or after y in Γ^+, respectively. ABLR-relations "B" and "R" can be derived as the inverse of "A" and "L". Examples are given in Fig. 1.

A remarkable feature of Sequence-Pair is that its generation and decoding are both possible by simple operations. The question is what the solution space of all SP's is

Theorem 3 *Any feasible placement has a corresponding SP that generates an ABLR-relation set satisfied by the placement. On the other hand, any SP has a corresponding placement that satisfies the ABLR-relation set derived from the SP.*

Using SP, a common compaction technique mentioned before is described in a very simple way:

Minimum Area Placement from $SP = (\Gamma^+, \Gamma^-)$

1. Relabel the objects such that $\Gamma^- = (1, 2, \ldots, n)$. Then $\Gamma^+ = (p_1, p_2, \ldots, p_n)$ will be a permutation of numbers $1, 2, \ldots, n$. It is simply a kind of normalization of *SP* [10]. But Kajitani [11] considers it a concept derived from Q-sequence [9] and studies its implication by the name of *Single-Sequence* or *SS*. In the example in Fig. 1b, p, q, r, and s are labeled as 1, 2, 3, and 4 so that $SS = (2413)$.
2. Take object 1 and place it at the left-bottom corner in the 1st quadrant.
3. For $k = 2, 3, \ldots, n$, place k such that its left edge is at the rightmost edge of the objects with smaller numbers than k and lie before k in *SS*, and its bottom edge is at the topmost edge of the objects with smaller numbers than k and lie after k in *SS*.

Applications

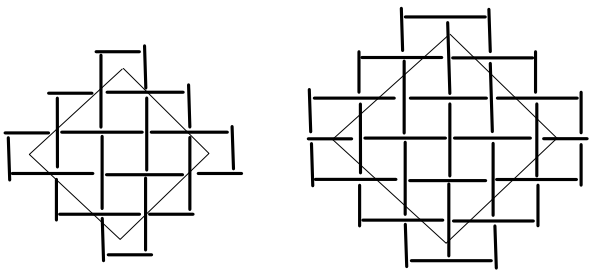
Many ideas followed after BSG and SP [2,3,4,5] as seen in the reference. They all applied a common methodology of a stochastic heuristic search, called Simulated Annealing, to generate feasible placements one after another based on some evaluation (with respect to the smallness of the bb-area), and to keep the best-so-far as the output. This methodology has become practical by the speed achieved due to their simple data structure. The first and naive implementation of BSG [2] could output the layout of sufficiently small area placement of five hundred rectangles in several minutes. (Finding a placement with the minimum bb-area is NP-hard [3].) Since then many ideas followed, including currently widely used codes such as O-tree [6], B*-tree [8], Corner–Block–List [7], Q-sequence [9], Single-Sequence [11], and others. Their common feature is in coding the nonoverlapping constraint along horizontal and vertical directions, which is the inheritant property of rectangles.

As long as applications are concerned with the rectangle placement in the minimum area, and do not mind mutual interconnection, the problem can be solved practically enough by BSG, SP, and those related ideas. However, in an integrated circuit layout problem, mutual connection is a major concern. Objects are not restricted to rectangles, even soft objects are used for performance. Many efforts have been devoted with a certain degree of success. For example, techniques concerned with rectilinear objects, rectilinear chip, insertion of small but numerous elements like buffers and decoupling capacitors, replacement for design change, symmetric placement for analog circuit design, 3-dimensional placement, etc. have been developed. Here few of them is cited but it is recommended to look at proceedings of ICCAD, DAC, ASPDAC, DATE, and journals TCAD, TCAS, particularly those that cover VLSI physical design.

Open Problems

BSG

The claim of Theorem 2 that a BSG needs n rows to provide any feasible ABLR-relation set is reasonable if considering a placement of all objects aligned vertically. This is due to the rectangular framework of a BSG. However, experiments have been suggesting a question if from the beginning [5] if we need such big BSGs. The octagonal BSG is defined in Fig. 4. It is believed to hold the following claim expecting a drastic reduction of the solution space.



Floorplan and Placement, Figure 4
Octagonal BSG of size *n*, *p*-type: a If *n* is odd, it has $(n^2 + 1)/2$ rooms. b If *n* is even, it has $(n^2 + 2n)/2$ rooms

Conjecture (BSG): For any feasible ABLR-relation set, there is an assignment of n objects into octagonal BSG of size n, any type, that generates the same ABLR-relation set.

If this is true, then the size of the solution space needed by a BSG reduces to ${}_{(n^2+1)/2}C_n$ or ${}_{(n^2+2n)/2}C_n$.

SP or SS

It is possible to define the universality of SP or SS in the same manner as defined for BSG. In general, two sequences of arbitrary k numbers $P = (p_1, p_2, \ldots, p_k)$ and Q=$(q_1, q_2, \ldots, q_k)$ are said *similar* with each other if ord(p_i) = ord(q_i) for every i where $ord(p_i) = j$ implies that p_i is the jth smallest in the sequence. If they are single-sequences, two similar sequences generate the same set of ABLR-relations under the natural one-to-one correspondence between numbers.

An *SS* of length m (necessarily $\geq n$) is said *universal of order n* if *SS* has a subsequence (a sequence obtained from *SS* by deleting some of the numbers) that is similar to any sequence of length n. Since rooms of a BSG are considered n^2 objects, Theorem 2 implies that there is a universal *SS* of order n whose length is n^2. The known facts about smaller universal *SS* are:

1. For $n = 2, 132, 231, 213$, and 312 are the shortest universal *SS*. Note that 123 and 321 are not universal.
2. For $n = 3$, $SS = 41352$ is the shortest universal *SP*.
3. For $n = 4$, the shortest length of universal *SS* 10 or less.
4. The size of universal *SS* is $\Omega(n^2)$ [12].

Open Problem (SP)

It is still an open problem to characterize the universal *SP*. For example, give a way to 1) certify a sequence as universal and 2) generate a minimum universal sequence for general n.

Cross References

- ▶ Bin Packing
- ▶ Circuit Placement
- ▶ Slicing Floorplan Orientation
- ▶ Sphere Packing Problem

Recommended Reading

1. Wong, D.F., Liu, C.L.: A new algorithm for floorplan design. In: ACM/IEEE Design Automation Conference (DAC), November 1985, 23rd, pp. 101–107
2. Nakatake, S., Murata, H., Fujiyoshi, K., Kajitani, Y.: Bounded Sliceline Grid (BSG) for module packing. IEICE Technical Report, October 1994, VLD94-66, vol. 94, no. 313, pp. 19–24 (in Japanese)
3. Murata, H., Fujiyoshi, K., Nakatake, S., Kajitani, Y.: A solution space of size $(n!)^2$ for optimal rectangle packing. In: 8th Karuizawa Workshop on Circuits and Systems, April 1995, pp. 109–114
4. Murata, H., Nakatake, S., Fujiyoshi, K., Kajitani, Y.: VLSI Module placement based on rectangle-packing by Sequence-Pair. IEEE Trans. Comput. Aided Design (TCAD) **15**(12), 1518–1524 (1996)
5. Nakatake, S., Fujiyoshi, K., Murata, H., Kajitani, Y.: Module packing based on the BSG-structure and IC layout applications. IEEE TCAD **17**(6), 519–530 (1998)
6. Guo, P.N., Cheng, C.K., Yoshimura, T.: An O-tree representation of non-slicing floorplan and its applications. In: 36th DAC., June 1998, pp. 268–273
7. Hong, X., Dong, S., Ma, Y., Cai, Y., Cheng, C.K., Gu, J.: Corner Block List: An efficient topological representation of non-slicing floorplan. In: International Computer Aided Design (ICCAD) '00, November 2000, pp. 8–12,
8. Chang, Y.-C., Chang, Y.-W., Wu, G.-M., Wu, S.-W.: B*-trees: A new representation for non-slicing floorplans. In: 37th DAC, June 2000, pp. 458–463
9. Sakanushi, K., Kajitani, Y., Mehta, D.: The quarter-state-sequence floorplan representation. In: IEEE TCAS-I: **50**(3), 376–386 (2003)
10. Kodama, C., Fujiyoshi, K.: Selected Sequence-Pair: An efficient decodable packing representation in linear time using Sequence-Pair. In: Proc. ASP-DAC 2003, pp. 331–337
11. Kajitani, Y.: Theory of placement by Single-Sequence Realted with DAG, SP, BSG, and O-tree. In: International Symposium on Circuts and Systems, May 2006
12. Imahori, S.: Privatre communication, December 2005

Flow Time Minimization

2001; Becchetti, Leonardi, Marchetti-Spaccamela, Pruhs

Luca Becchetti[1], Stefano Leonardi[1], Alberto Marchetti-Spaccamela[1], Kirk Pruhs[2]

[1] Department of Information and Computer Systems, University of Rome, Rome, Italy

[2] Computer Science, University of Pittsburgh, Pittsburgh, PA, USA

Keywords and Synonyms

Flow time: response time

Problem Definition

Shortest-job-first heuristics arise in sequencing problems, when the goal is minimizing the perceived latency of users of a multiuser or multitasking system. In this problem, the algorithm has to schedule a set of jobs on a pool of m identical machines. Each job has a release date and a processing time, and the goal is to minimize the average time spent by jobs in the system. This is normally considered a suitable measure of the quality of service provided by a system to

interactive users. This optimization problem can be more formally described as follows:

Input A set of m identical machines and a set of n jobs $1, 2, \ldots, n$. Every job j has a release date r_j and a processing time p_j. In the sequel, $\mathcal{I}$ denotes the set of feasible input instances.

Goal The goal is minimizing the *average flow* (also known as *average response*) time of the jobs. Let C_j denote the time at which job j is completed by the system. The flow time or response time F_j of job j is defined by $F_j = C_j - r_j$. The goal is thus minimizing

$$\min \frac{1}{n} \sum_{j=1}^{n} F_j \,.$$

Since n is part of the input, this is equivalent to minimizing the *total* flow time, i. e. $\sum_{j=1}^{n} F_j$.

Off-line versus on-line In the *off-line setting*, the algorithm has full knowledge of the input instance. In particular, for every $j = 1, \ldots, n$, the algorithm knows r_j and p_j.

Conversely, in the *on-line setting*, at any time t, the algorithm is only aware of the set of jobs released up to time t.

In the sequel, A and OPT denote, respectively, the algorithm under consideration and the optimal, off-line policy for the problem. $A(I)$ and $OPT(I)$ denote the respective costs on a specific input instance I.

Further assumptions in the on-line case Further assumptions can be made as to the algorithm's knowledge of processing times of jobs. In particular, in this survey an important case is considered, realistic in many applications, i. e. that p_j is completely unknown to the on-line algorithms until the job eventually completes (*non-clairvoyance*) [1,3].

Performance metric In all cases, as is common in combinatorial optimization, the performance of the algorithm is measured with respect to its optimal, off-line counterpart. In a minimization problem such as those considered in this survey, the competitive ratio ρ_A is defined as:

$$\rho_A = \max_{I \in \mathcal{I}} \frac{A(I)}{OPT(I)} \,.$$

In the off-line case, ρ_A is the *approximation ratio* of the algorithm. In the on-line setting, ρ_A is known as the *competitive ratio* of A.

Preemption When *preemption* is allowed, a job that is being processed may be interrupted and resumed later after processing other jobs in the interim. As shown further, preemption is necessary to design efficient algorithms in the framework considered in this survey [5,6].

Key Results

Algorithms

Consider any job j in the instance and a time t in A's schedule, and denote by $w_j(t)$ the amount of time spent by A on job j until t. Denote by $x_j(t) = p_j - w_j(t)$ its *remaining processing time* at t.

The best known heuristic for minimizing the average flow time when preemption is allowed is *shortest remaining processing time* (SRPT). At any time t, SRPT executes a pending job j such that $x_j(t)$ is minimum. When preemption is not allowed, this heuristic translates to *shortest job first* (SJF): at the beginning of the schedule, or when a job completes, the algorithm chooses a pending job with the shortest processing time and runs it to completion.

Complexity

The problem under consideration is polynomially solvable on a single machine when preemption is allowed [9,10]. When preemption is allowed, SRPT is optimal for the single-machine case. On parallel machines, the best known upper bound for the preemptive case is achieved by SRPT, which was proven to be $O(\log \min n/m, P)$-approximate [6], P being the ratio between the largest and smallest processing times of the instance. Notice that SRPT is an on-line algorithm, so the previous result holds for the on-line case as well. The authors of [6] also prove that this lower bound is tight in the on-line case. In the off-line case, no non-constant lower bound is known when preemption is allowed.

In the non-preemptive case, no off-line algorithm can be better than $\Omega(n^{1/3-\epsilon})$-approximate, for every $\epsilon > 0$, the best upper bound being $O(\sqrt{n/m}\log(n/m))$ [6]. The upper and lower bound become $O(\sqrt{n})$ and $\Omega(n^{1/2-\epsilon})$ for the single machine case [5].

Extensions Many extensions have been proposed to the scenarios described above, in particular for the preemptive, on-line case. Most proposals concern the power of the algorithm or the knowledge of the input instance. For the former aspect, one interesting case is the one in which the algorithm is equipped with faster machines than its optimal counterpart. This aspect has been considered in [4]. There the authors prove that even a moderate increase

in speed makes some very simple heuristics have performances that can be very close to the optimum.

As to the algorithm's knowledge of the input instance, an interesting case in the on-line setting, consistent with many real applications, is the non-clairvoyant case described above. This aspect has been considered in [1,3]. In particular, the authors of [1] proved that a randomized variant of the MLF heuristic described above achieves a competitive ratio that in the average is at most a polylogarithmic factor away from the optimum.

Applications

The first and traditional field of application for scheduling policies is resource assignment to processes in multitasking operating systems [11]. In particular, the use of shortest-job-like heuristics, notably the MLF heuristic, is documented in operating systems of wide use, such as UNIX and WINDOWS NT [8,11]. Their application to other domains, such as access to Web resources, has been considered more recently [2].

Open Problems

Shortest-job-first-based heuristics such as those considered in this survey have been studied in depth in the recent past. Still, some questions remain open. One concerns the off-line, parallel-machine case, where no non-constant lower bound on the approximation is known yet. As to the on-line case, there still is no tight lower bound for the non-clairvoyant case on parallel machines. The current $\Omega(\log n)$ lower bound was achieved for the single-machine case [7], and there are reasons to believe that it is below the one for the parallel case by a logarithmic factor.

Cross References

- ► Minimum Flow Time
- ► Minimum Weighted Completion Time
- ► Multi-level Feedback Queues
- ► Shortest Elapsed Time First Scheduling

Recommended Reading

1. Becchetti, L., Leonardi, S.: Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. J. ACM **51**(4), 517–539 (2004)
2. Crovella, M.E., Frangioso, R., Harchal-Balter, M.: Connection scheduling in web servers. In: Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems (USITS-99), 1999 pp. 243–254
3. Kalyanasundaram, B., Pruhs, K.: Minimizing flow time nonclairvoyantly. J. ACM **50**(4), 551–567 (2003)
4. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM **47**(4), 617–643 (2000)
5. Kellerer, H., Tautenhahn, T., Woeginger, G.J.: Approximability and nonapproximability results for minimizing total flow time on a single machine. In: Proceedings of 28th Annual ACM Symposium on the Theory of Computing (STOC '96), 1996, pp. 418–426
6. Leonardi, S., Raz, D.: Approximating total flow time on parallel machines. In: Proceedings of the Annual ACM Symposium on the Theory of Computing STOC, 1997, pp. 110–119
7. Motwani, R., Phillips, S., Torng, E.: Nonclairvoyant scheduling. Theor. Comput. Sci. **130**(1), 17–47 (1994)
8. Nutt, G.: Operating System Projects Using Windows NT. Addison-Wesley, Reading (1999)
9. Schrage, L.: A proof of the optimality of the shortest remaining processing time discipline. Oper. Res. **16**(1), 687–690 (1968)
10. Smith, D.R.: A new proof of the optimality of the shortest remaining processing time discipline. Oper. Res. **26**(1), 197–199 (1976)
11. Tanenbaum, A.S.: Modern Operating Systems. Prentice-Hall, Englewood Cliffs (1992)

Formal Methods

- ► Learning Automata
- ► Symbolic Model Checking

FPGA Technology Mapping

1992; Cong, Ding

Jason Cong[1], Yuzheng Ding[2]

[1] Department of Computer Science, UCLA, Los Angeles, CA, USA

[2] Synopsys Inc., Mountain View, CA, USA

Keywords and Synonyms

Lookup-Table Mapping; LUT Mapping; FlowMap

Problem Definition

Introduction

Field Programmable Gate Array (FPGA) is a type of integrated circuit (IC) device that can be (re)programmed to implement custom logic functions. A majority of FPGA devices use lookup-table (LUT) as the basic logic element, where a LUT of K logic inputs (K-LUT) can implement any Boolean function of up to K variables. An FPGA also contains other logic elements, such as registers, programmable interconnect resources, and input/output resources [5].

The programming of an FPGA involves the transformation of a logic design into a form suitable for implementation on the target FPGA device. This generally takes multiple steps. For LUT based FPGAs, *technology mapping* is to transform a general Boolean logic network (obtained from the design specification through earlier transformations) into a functional equivalent K-LUT network that can be implemented by the target FPGA device. The objective of a technology mapping algorithm is to generate, among many possible solutions, an optimized one according to certain criteria, some of which are: timing optimization, which is to make the resultant implementation operable at faster speed; area minimization, which is to make the resultant implementation compact in size; power minimization, which is to make the resultant implementation low in power consumption. The algorithm presented here, named *FlowMap* [2], is for timing optimization; it was the first provably optimal polynomial time algorithm for technology mapping problems on general Boolean networks, and the concepts and approach it introduced has since generated numerous useful derivations and applications.

Data Representation and Preliminaries

The input data to a technology mapping algorithm for LUT based FPGA is a *general Boolean network*, which can be modeled as a direct acyclic graph $N = (V, E)$. A node $v \in V$ can either represent a logic signal source from outside of the network, in which case it has no incoming edge and is called a *primary input* (PI) node; or it can represent a *logic gate*, in which case it has incoming edge(s) from PIs and/or other gates, which are its logic input(s). If the logic output of the gate is also used outside of the network, its node is a primary output (PO), which can have no outgoing edge if it is only used outside.

If $\langle u, v\rangle \in E$, u is said to be a *fanin* of v, and v a *fanout* of u. For a node v, $input(v)$ denotes the set of its fanins; similarly for a subgraph H, $input(H)$ denotes the set of distinct nodes outside of H that are fanins of nodes in H. If there is a direct path in N from a node u to a node v, u is said to be a *predecessor* of v and v a *successor* of u. The *input network* of a node v, denoted N_v, is the subgraph containing v and all of its predecessors. A *cone* of a non-PI node v, denoted C_v, is a subgraph of N_v containing v and possibly some of its *non-PI* predecessors, such that for any node $u \in C_v$, there is a path from u to v in C_v. If $|input(C_v)| \leq K$, C_v is called a *K-feasible* cone. The network N is *K-bounded* if every non-PI node has a K-feasible cone. A *cut* of a non-PI node v is a bipartition (X, X') of nodes in N_v such that X' is a cone of v; $input(X')$ is called the *cut-set* of (X, X'), and $n(X, X') = |input(X')|$ the *size* of the cut. If $n(X, X') \leq K$, (X, X') is a *K-feasible* cut. The *volume* of (X, X') is $vol(X, X') = |X'|$.

A *topological order* of the nodes in the network N is a linear ordering of the nodes in which each node appears after all of its predecessors and before any of its successors. Such an order is always possible for an acyclic graph.

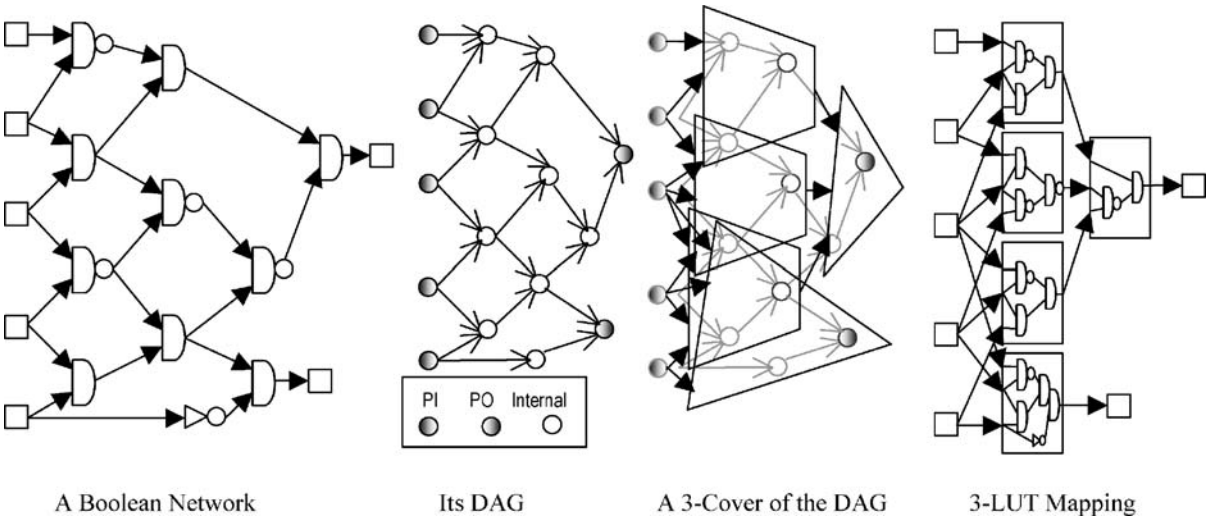
Problem Formulation

A *K-cover* of a given Boolean network N is a network $N_M = (V_M, E_M)$, where V_M consists of the PI nodes of N and some K-feasible cones of nodes in N, such that for each PO node v of N, V_M contains a cone C_v of v; and if $C_u \in V_M$, then for each non-PI node $v \in input(C_u)$, V_M also contains a cone C_v of v. edge $\langle u, C_v\rangle \in E_M$ if and only if PI node $u \in input(C_v)$; edge $\langle C_u, C_v\rangle \in E_M$ if and only if non-PI node $u \in input(C_v)$. Since each K-feasible cone can be implemented by a K-LUT, a K-cover can be implemented by a network of K-LUTs. Therefore, the *technology mapping* problem for K-LUT based FPGA, which is to transform N into a network of K-LUTs, is to find a K-cover N_M of N.

The *depth* of a network is the number of edges in its longest path. A technology mapping solution N_M is *depth optimal* if among all possible mapping solutions of N it has the minimum depth. If each level of K-LUT logic is assumed to contribute a constant amount of logic delay (known as the *unit delay* model), the minimum depth corresponds to the smallest logic propagation delay through the mapping solution, or in other words, the fastest K-LUT implementation of the network N. The problem solved by the *FlowMap* algorithm is *depth optimal technology mapping for K-LUT based FPGAs*.

A Boolean network that is not K-bounded may not have a mapping solution as defined above. To make a network K-bounded, *gate decomposition* may be used to break larger gates into smaller ones. The *FlowMap* algorithm applies, as pre-processing, an algorithm named *DMIG* [3] that converts all gates into 2-input ones in a depth optimal fashion, thus making the network K-bounded for $K \geq 2$. Different decomposition schemes may result in different K-bounded networks, and consequently different mapping solutions; the optimality of *FlowMap* is with respect to a given K-bounded network.

Figure 1 illustrates a Boolean network, its DAG, a covering with 3-feasible cones, and the resultant 3-LUT network. As illustrated, the cones in the covering may overlap; this is allowed and often beneficial. (When the mapped network is implemented, the overlapped portion of logic will be replicated into each of the K-LUTs that contain it.)



FPGA Technology Mapping, Figure 1

Key Results

The *FlowMap* algorithm takes a two-phase approach. In the first phase, it determines for each non-PI node a preferred K-feasible cone as a candidate for the covering; the cones are computed such that if used, they will yield a depth optimal mapping solution. This is the central piece of the algorithm. In the second phase the cones necessary to form a cover are chosen to generate a mapping solution.

Structure of Depth Optimal K-covers

Let $M(v)$ denote a K-cover (or equivalently, K-LUT mapping solution) of the input network N_v of v. If v is a PI, $M(v)$ consists of v itself. (For simplicity, in the rest of the article $M(v)$ shall be referred as *a K-cover of v*.) With that defined, first there is

Lemma 1 *If C_v is the K-feasible cone of v in a K-cover $M(v)$, then $M(v) = \{C_v\} + \bigcup\{M(u): u \in input(C_v)\}$ where $M(u)$ is a certain K-cover of u. Conversely, if C_v is a K-feasible cone of v, and for each $u \in input(C_v)$, $M(u)$ a K-cover of u, then $M(v) = \{C_v\} + \bigcup\{M(u) : u \in input(C_v)\}$ is a K-cover of v.*

In other words, a K-cover consists of a K-feasible cone and a K-cover of each input of the cone. Note that for $u_1 \in input(C_v)$, $u_2 \in input(C_v)$, $M(u_1)$ and $M(u_2)$ may overlap, and an overlapped portion may or may not be covered the same way; the union above includes all *distinct* cones from all parts. Also note that for a given C_v, there can be different K-covers of v containing C_v, varying by the choice of $M(u)$ for each $u \in input(C_v)$.

Let $d(M(v))$ denote the depth of $M(v)$. Then

Lemma 2 *For K-cover $M(v) = \{C_v\} + \bigcup\{M(u) : u \in input(C_v)\}$, $d(M(v)) = \max\{d(M(u)) : u \in input(C_v)\}+1$.*

In particular, let $M^*(u)$ denote a K-cover of u with minimum depth, then $d(M(v)) \geq \max\{d(M^*(u)) : u \in input(C_v)\}+1$; the equality holds when every $M(u)$ in $M(v)$ is of minimum depth.

Recall that C_v defines a K-feasible cut (X, X') where $X' = C_v$, $X = N_v - C_v$. Let $H(X, X')$ denote the *height* of the cut (X, X'), defined as $H(X, X') = \max\{d(M^*(u)): u \in input(X')\} + 1$. Clearly, $H(X, X')$ gives the minimum depth of any K-cover of v containing $C_v = X'$. Moreover, by properly choosing the cut, $H(X, X')$ height can be minimized, which leads to a K-cover with minimum depth:

Theorem 1 *If K-feasible cut (X, X') of v has the minimum height among all K-feasible cuts of v, then the K-cover $M^*(v) = \{X'\} + \bigcup\{M^*(u) : u \in input(X')\}$, is of minimum depth among all K-covers of v.*

That is, a minimum height K-feasible cut defines a minimum depth K-cover. So the central task for depth optimal technology mapping becomes the computation of a minimum height K-feasible cut for each PO node.

By definition, the height of a cut depends on the (depths of) minimum depth K-covers of nodes in $N_v - \{v\}$. This suggests a *dynamic programming* procedure that follows topological order, so that when the minimum depth K-cover of v is to be determined, a minimum depth K-cover of each node in $N_v - \{v\}$ is already known and the height of a cut can be readily determined. This is how the first phase of the *FlowMap* algorithm is carried out.

Minimum Height *K*-feasible Cut Computation

The first phase of *FlowMap* was originally called the *labeling phase*, as it involves the computation of a *label* for each node in the *K*-bounded graph. The label of a non-PI node v, denoted $l(v)$, is defined as the minimum height of any cut of v. For convenience, the labels of PI nodes are defined to be 0.

The so defined label has an important *monotonic* property.

Lemma 3 *Let* $p = \max\{l(u) : u \in input(v)\}$, *then* $p \leq l(v) \leq p + 1$.

Note that this also implies that for any node $u \in N_v - \{v\}$, $l(u) \leq p$. Based on this, in order to find a minimum height *K*-feasible cut, it is sufficient to check if there is one of height p; if not, then any *K*-feasible cut will be of minimum height $(p + 1)$, and one always exists for a *K*-bounded graph.

The search for a K-feasible cut of a height p ($p > 0$; $p = 0$ is trivial) in *FlowMap* is done by transforming N_v into a *flow network* F_v and computing a *network flow* [4] on it (hence the name). The transformation is as follows. For each node $u \in N_v - \{v\}$, $l(u) < p$, F_v has two nodes u_1 and u_2, linked by a *bridge* edge $\langle u_1, u_2\rangle$; F_v has a single *sink* node t for all other nodes in N_v, and a single *source* node s. For each PI node u of N_v, which corresponds to a bridge edge $\langle u_1, u_2\rangle$ in F_v, F_v contains edge $\langle s, u_1\rangle$; for each edge $\langle u, w\rangle$ in N_v, if both u and w have bridge edges in F_v, then F_v contains edge $\langle u_2, w_1\rangle$; if u has a bridge edge but w does not, F_v contains edge $\langle u_2, t\rangle$; otherwise (neither has bridge) no corresponding edge is in F_v. The bridging edges have unit capacity; all others have infinite capacity. Noting that each edge in F_v with finite (unit) capacity corresponds to a node $u \in N_v$ with $l(u) < p$ and vice versa, and according to the Max-Flow Min-Cut Theorem [4], it can be shown

Lemma 4 *Node v has a K-feasible cut of height p if and only if* F_v *has a maximum network flow of size no more than K.*

On the flow network F_v, a maximum flow can be computed by running the augmenting path algorithm [4]. Once a maximum flow is obtained, the *residual graph* of the flow network is disconnected, and the corresponding *min-cut* (X, X') can be identified as follows: $v \in X'$; for $u \in N_v - \{v\}$, if it is bridged in F_v, and u_1 can be reached in a depth-first search of the residual graph from s, then $u \in X$; otherwise $u \in X'$.

Note that as soon as the flow size exceeds K, the computation can stop, knowing there will not be a desired *K*-feasible cut. In this case, one can modify the flow network by bridging all node in $N_v - \{v\}$ allowing the inclusion of nodes u with $l(u) = p$ in the cut computation, and find a *K*-feasible cut with height p+1 the same way.

An augmenting path is found in linear time to the number of edges, and there are at most K augmentations for each cut computation. Applying the algorithm to every node in topological order, one would have

Theorem 2 *In a K-bounded Boolean network of n nodes and m edges, the computation of a minimum height K-feasible cut for every node can be completed in O(Kmn) time.*

The cut found by the algorithm has another property:

Lemma 5 *The cut* (X, X') *computed as above is the unique maximum volume min-cut; moreover, if* (Y, Y') *is another min-cut, then* $Y' \subseteq X'$.

Intuitively a cut of larger volume defines a larger cone which covers more logic, therefore a cut of larger volume is preferred. Note however Lemma 5 only claims maximum among min-cuts; if $n(X, X') < K$, there can be other cuts that are still *K*-feasible, but with larger cut size and larger cut volume. A post-processing algorithm used by *FlowMap* tries to grow (X, X') by collapsing all nodes in X', plus one or more in the cut-set, into the sink, and repeat the flow computation; this will force a cut of larger volume, an improvement if it is still *K*-feasible.

K-cover Construction

Once minimum height K-feasible cuts have been computed for all nodes, each node v has a *K*-feasible cone C_v defined by its cut, which has minimum depth. From here, constructing the *K*-cover $N_M = (V_M, E_M)$ is straightforward. First, the cones of all PO nodes are included in V_M. Then, for any cone $C_v \in V_M$, cone C_u for each non-PI node $u \in input(v)$ is also include in V_M; so is every PI node $u \in input(v)$. Similarly, an $\langle C_u, C_v\rangle \in E_M$ for each non-PI node $u \in input(C_v)$; $\langle u, C_v\rangle \in E_M$ for each PI node $u \in input(C_v)$.

Lemma 6 *The K-cover constructed as above is depth optimal.*

This is a linear time procedure, therefore

Theorem 3 *The problem of depth optimal technology mapping for K-LUT based FPGAs on a Boolean network of n nodes and m edges can be solved in O(Kmn) time.*

Applications

The *FlowMap* algorithm has been used as a center piece or a framework for more complicated FPGA logic synthesis

and technology mapping algorithms. There are many possible variations that can address various needs in its applications. Some are briefed below; details of such variations/applications can be found in [1,3].

Complicated Delay Models

With minimal change the algorithm can be applied where non-unit delay model is used, allowing delay of the nodes and/or the edges to vary, as long as they are static. Dynamic delay models, where the delay of a net is determined by its post-mapping structure, cannot be applied to the algorithm; In fact, delay optimal mapping under dynamic delay models is NP-hard [3].

Complicated Architectures

The algorithm can be adapted to FPGA architectures that are more sophisticated than homogeneous K-LUT arrays. For example, mapping for FPGA with two LUT sizes can be carried out by computing a cone for each size and dynamically choosing the best one.

Multiple Optimization Objectives

While the algorithm is for delay minimization, area minimization (in terms of the number of cones selected) as well as other objectives can also be incorporated, by adapting the criteria for cut selection. The original algorithm considers area minimization by maximizing the volume of the cuts; substantially more minimization can be achieved by considering more K-feasible cuts, and make smart choices to e. g. increase sharing among input networks, allow cuts of larger heights along no-critical paths, etc. Achieving area optimality, however, is NP-hard.

Integration with Other Optimizations

The algorithm can be combined with other types of optimizations, including retiming, logic resynthesis, and physical synthesis.

Cross References

- ▶ Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach
- ▶ Performance-Driven Clustering
- ▶ Sequential Circuit Technology Mapping

Recommended Reading

The *FlowMap* algorithm, with more details and experimental results, was published in [2]. General information about FPGA can be found in [5]. A good source of concepts and algorithms of network flow is [4]. Comprehensive surveys of FPGA design automation, including many variations and applications of the *FlowMap* algorithm, as well as other algorithms, are presented in [1,3].

1. Chen, D., Cong, J., Pan, P.: FPGA design automation: a survey. Foundations and Trends in Electronic Design Automation, vol 1, no 3. Now Publishers, Hanover, USA (2006)
2. Cong, J., Ding, Y.: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs, Proc. IEEE/ACM International Conference on Computer-Aided Design, pp. 48–53. San Jose, USA (1992)
3. Cong, J., Ding, Y.: Combinational logic synthesis for LUT based field programmable gate arrays. ACM Trans. Design Autom. Electron. Sys. **1**(2): 145–204 (1996)
4. Tarjan, R.: Data Structures and Network Algorithms. SIAM. Philadelphia, USA (1983)
5. Trimberger, S.: Field-Programmable Gate Array Technology. Springer, Boston, USA (1994)

Fractional Packing and Covering Problems

1991; Plotkin, Shmoys, Tardos
1995; Plotkin, Shmoys, Tardos

George Karakostas
Department of Computing & Software,
McMaster University, Hamilton, ON, Canada

Problem Definition

This entry presents results on fast algorithms that produce approximate solutions to problems which can be formulated as Linear Programs (LP), and therefore can be solved exactly, albeit with slower running times. The general format of the family of these problems is the following: Given a set of m inequalities on n variables, and an oracle that produces the solution of an appropriate optimization problem over a convex set $P \in \mathbb{R}^n$, find a solution $x \in P$ that satisfies the inequalities, or detect that no such x exists. The basic idea of the algorithm will always be to start from an infeasible solution x, and use the optimization oracle to find a direction in which the violation of the inequalities can be decreased; this is done by calculating a vector y that is a *dual solution corresponding to* x. Then x is carefully updated towards that direction, and the process is repeated until x becomes 'approximately' feasible. In what follows, the particular problems tackled, together with the corresponding optimization oracle, as well as the different notions of 'approximation' used are defined.

- The **fractional packing problem** and its oracle are defined as follows:

  PACKING: Given an $m \times n$ matrix A, $b > 0$, and a convex set P in $\mathbb{R}^n$ such that $Ax \geq 0$, $\forall x \in P$, is there $x \in P$ such that $Ax \leq$ b?

  PACK_ORACLE: Given m-dimensional vector $y \geq 0$ and P as above, return $\bar{x} := \arg\min\{y^{\mathrm{T}}Ax : x \in P\}$.

- The **relaxed fractional packing problem** and its oracle are defined as follows:

  RELAXED PACKING: Given $\varepsilon > 0$, an $m \times n$ matrix A, $b > 0$, and convex sets P and $\hat{P}$ in $\mathbb{R}^n$ such that $P \subseteq \hat{P}$ and $Ax \geq 0$, $\forall x \in \hat{P}$, find $x \in \hat{P}$ such that $Ax \leq (1+\varepsilon)b$, or show that $\nexists x \in P$ such that $Ax \leq$ b.

  REL_PACK_ORACLE: Given m-dimensional vector $y \geq 0$ and $P, \hat{P}$ as above, return $\bar{x} \in \hat{P}$ such that $y^{\mathrm{T}}A\bar{x} \leq \min\{y^{\mathrm{T}}Ax : x \in P\}$.

- The **fractional covering problem** and its oracle are defined as follows:

  COVERING: Given an $m \times n$ matrix A, $b > 0$, and a convex set P in $\mathbb{R}^n$ such that $Ax \geq 0$, $\forall x \in P$, is there $x \in P$ such that $Ax \geq b$?

  COVER_ORACLE: Given m-dimensional vector $y \geq 0$ and P as above, return $\bar{x} := \arg\max\{y^{\mathrm{T}}Ax : x \in P\}$.

- The **simultaneous packing and covering problem** and its oracle are defined as follows:

  SIMULTANEOUS PACKING AND COVERING: Given $\hat{m} \times n$ and $(m - \hat{m}) \times n$ matrices $\hat{A}, A$ respectively, $b > 0$ and $\hat{b} > 0$, and a convex set P in $\mathbb{R}^n$ such that $Ax \geq 0$ and$\hat{A}x \geq 0$, $\forall x \in P$, is there $x \in P$ such that $Ax \leq$ b, and $\hat{A}x \geq \hat{b}$?

  SIM_ORACLE: Given P as above, a constant ν and a dual solution $(y, \hat{y})$, return $\bar{x} \in P$ such that

  $$A\bar{x} \leq \nu b, \text{ and}$$
  $$y^{\mathrm{T}}A\bar{x} - \sum_{i \in I(\nu,\bar{x})} \hat{y}_i \hat{a}_i \bar{x} = \min\{y^{\mathrm{T}}Ax - \sum_{i \in I(\nu,x)} \hat{y}_i \hat{a}_i x : x \text{ a vertex of } P \text{ such that } Ax \leq \nu b\},$$

  where $I(\nu, x) := \{i : \hat{a}_i x \leq \nu b_i\}$.

- The **general problem** and its oracle are defined as follows:

  GENERAL: Given an $m \times n$ matrix A, an arbitrary vector b, and a convex set P in $\mathbb{R}^n$, is there $x \in P$ such that $Ax \leq$ b,?

  GEN_ORACLE: Given m-dimensional vector $y \geq 0$ and P as above, return $\bar{x} := \arg\min\{y^{\mathrm{T}}Ax : x \in P\}$.

Definitions and Notation

For an error parameter $\varepsilon > x0$, a point $x \in P$ is an *ε-approximation solution* for the fractional packing (or covering) problem if $Ax \leq (1+\varepsilon)b$ (or $Ax \geq (1-\varepsilon)b$). On the other hand, if $x \in P$ satisfies $Ax \leq$ b (or $Ax \geq b$), then x is an *exact solution*. For the GENERAL problem, given an error parameter $\varepsilon > 0$ and a positive tolerance vector d, $x \in P$ is an *ε-approximation solution* if $Ax \leq b + \varepsilon d$, and an *exact solution* if $Ax \leq$ b. An *ε-relaxed decision procedure* for these problems either finds an ε-approximation solution, or correctly reports that no exact solution exists. In general, for a minimization (maximization) problem, an $(1+\varepsilon)$-approximation ($(1-\varepsilon)$-*approximation*) *algorithm* returns a solution at most $(1+\varepsilon)$ (at least $(1-\varepsilon)$) times the optimal.

The algorithms developed work within time that depends polynomially on ε^{-1}, for any error parameter $\varepsilon > 0$. Their running time will also depend on the *width* ρ of the convex set P relative to the set of inequalities $Ax \leq$ b or $Ax \geq b$ defining the problem at hand. More specifically the width ρ is defined as follows for each one of the problems considered here:

- PACKING: $\rho := \max_i \max_{x \in P} \frac{a_i x}{b_i}$.
- RELAXED PACKING: $\hat{\rho} := \max_i \max_{x \in \hat{P}} \frac{a_i x}{b_i}$.
- COVERING: $\rho := \max_i \max_{x \in P} \frac{a_i x}{b_i}$.
- SIMULTANEOUS PACKING AND COVERING: $\rho := \max_{x \in P} \max\{\max_i \frac{a_i x}{b_i}, \max_i \frac{\hat{a}_i x}{\hat{b}_i}\}$.
- GENERAL: $\rho := \max_i \max_{x \in P} \frac{|a_i x - b_i|}{d_i} + 1$, where d is the tolerance vector defined above.

Key Results

Many of the results below were presented in [7] by assuming a model of computation with exact arithmetic on real numbers and exponentiation in a single step. But, as the authors mention [7], they can be converted to run on the RAM model by using approximate exponentiation, a version of the oracle that produces a *nearly* optimal solution, and a limit on the numbers used that is polynomial in the input length similar to the size of numbers used in exact linear programming algorithms. However they leave as an open problem the construction of ε-approximate solutions using polylogarithmic precision for the general case of the problems they consider (as can be done, for example, in the multicommodity flow case [4]).

Theorem 1 *For $0 < \varepsilon \leq 1$, there is a deterministic ε-relaxed decision procedure for the fractional packing problem that uses $O(\varepsilon^{-2}\rho \log(m\varepsilon^{-1}))$ calls to* PACK_ORACLE, *plus*

the time to compute Ax for the current iterate x between consecutive calls.

For the case of P being written as a product of smaller-dimension polytopes, i. e., $P = P^1 \times \cdots \times P^k$, each P^l with width ρ^l (obviously $\rho \le \sum_l \rho^l$), and a separate PACK_ORACLE for each P^l, A^l, then randomization can be used to potentially speed up the algorithm. By using the notation PACK_ORACLE$_l$ for the P^l, A^l oracle, the following holds:

Theorem 2 *For $0 < \varepsilon \le 1$, there is a randomized ε-relaxed decision procedure for the fractional packing problem that is expected to use $O(\varepsilon^{-2}(\sum_l \rho^l)\log(m\varepsilon^{-1}) + k\log(\rho\varepsilon^{-1}))$ calls to* PACK_ORACLE$_l$ *for some $l \in \{1,\ldots,k\}$ (possibly a different l in every call), plus the time to compute $\sum_l A^l x^l$ for the current iterate $x = (x^1, x^2, \ldots, x^k)$ between consecutive calls.*

Theorem 2 holds for RELAXED PACKING as well, if ρ is replaced by $\hat{\rho}$ and PACK_ORACLE by REL_PACK_ORACLE.

In fact, one needs only an *approximate* version of PACK_ORACLE. Let $C_P(y)$ be the minimum cost $y^T Ax$ achieved by PACK_ORACLE for a given y.

Theorem 3 *Let* PACK_ORACLE *be replaced by an oracle that given vector $y \ge 0$, finds a point $\bar{x} \in P$ such that $y^T A\bar{x} \le (1+\varepsilon/2)C_P(y) + (\varepsilon/2)\lambda y^T b$, where λ is minimum so that $Ax \le \lambda b$ is satisfied by the current iterate x. Then Theorems 1 and 2 still hold.*

Theorem 3 shows that even if no efficient implementation exists for an oracle, as in, e. g., the case when this oracle solves an NP-hard problem, a fully polynomial approximation scheme for it suffices.

Similar results can be proven for the fractional covering problem (COVER_ORACLE$_l$ is defined similarly to PACK_ORACLE$_l$ above):

Theorem 4 *For $0 < \varepsilon < 1$, there is a deterministic ε-relaxed decision procedure for the fractional covering problem that uses $O(m + \rho\log^2 m + \varepsilon^{-2}\rho\log(m\varepsilon^{-1}))$ calls to* COVER_ORACLE*, plus the time to compute Ax for the current iterate x between consecutive calls.*

Theorem 5 *For $0 < \varepsilon < 1$, there is a randomized ε-relaxed decision procedure for the fractional packing problem that is expected to use $O(mk + (\sum_l \rho^l)\log^2 m + k\log\varepsilon^{-1} + \varepsilon^{-2}(\sum_l \rho^l)\log(m\varepsilon^{-1}))$ calls to* COVER_ORACLE$_l$ *for some $l \in \{1,\ldots,k\}$ (possibly a different l in every call), plus the time to compute $\sum_l A^l x^l$ for the current iterate $x = (x^1, x^2, \ldots, x^k)$ between consecutive calls.*

Let $C_C(y)$ be the maximum cost $y^T Ax$ achieved by COVER_ORACLE for a given y.

Theorem 6 *Let* COVER_ORACLE *be replaced by an oracle that given vector $y \ge 0$, finds a point $\bar{x} \in P$ such that $y^T A\bar{x} \ge (1-\varepsilon/2)C_C(y) - (\varepsilon/2)\lambda y^T b$, where λ is maximum so that $Ax \ge \lambda b$ is satisfied by the current iterate x. Then Theorems 4 and 5 still hold.*

For the simultaneous packing and covering problem, the following is proven:

Theorem 7 *For $0 < \varepsilon \le 1$, there is a randomized ε-relaxed decision procedure for the simultaneous packing and covering problem that is expected to use $O(m^2(\log^2\rho)\varepsilon^{-2}\log(\varepsilon^{-1}m\log\rho))$ calls to* SIM_ORACLE*, and a deterministic version that uses a factor of $\log\rho$ more calls, plus the time to compute $\hat{A}x$ for the current iterate x between consecutive calls.*

For the GENERAL problem, the following is shown:

Theorem 8 *For $0 < \varepsilon < 1$, there is a deterministic ε-relaxed decision procedure for the* GENERAL *problem that uses $O(\varepsilon^{-2}\rho^2\log(m\rho\varepsilon^{-1}))$ calls to* GEN_ORACLE*, plus the time to compute Ax for the current iterate x between consecutive calls.*

The running times of these algorithms are proportional to the width ρ, and the authors devise techniques to reduce this width for many special cases of the problems considered. One example of the results obtained by these techniques is the following: If a packing problem is defined by a convex set that is a product of k smaller-dimension convex sets, i. e., $P = P^1 \times \cdots \times P^k$, and the inequalities $\sum_l A^l x^l \le b$, then there is a randomized ε-relaxed decision procedure that is expected to use $O(\varepsilon^{-2}k\log(m\varepsilon^{-1}) + k\log k)$ calls to a subroutine that finds a minimum-cost point in $\hat{P}^l = \{x^l \in P^l : A^l x^l \le b\}, l = 1,\ldots,k$, and a deterministic version that uses $O(\varepsilon^{-2}k^2\log(m\varepsilon^{-1}))$ such calls, plus the time to compute Ax for the current iterate x between consecutive calls. This result can be applied to the multicommodity flow problem, but the required subroutine is a single-source minimum-cost flow computation, instead of a shortest-path calculation needed for the original algorithm.

Applications

The results presented above can be used in order to obtain fast approximate solutions to linear programs, even if these can be solved exactly by LP algorithms. Many approximation algorithms are based on the rounding of the solution of such programs, and hence one might want to solve them approximately (with the overall approximation factor absorbing the LP solution approximation fac-

tor), but more efficiently. Two such examples, that appear in [7], are mentioned here.

Theorems 1, 2 can be applied for the improvement of the running time of the algorithm by Lenstra, Shmoys, and Tardos [5] for the scheduling of unrelated parallel machines without preemption ($R||C_{\max}$): N jobs are to be scheduled on M machines, with each job i scheduled on exactly one machine j with processing time p_{ij}, so that the maximum total processing time over all machines is minimized. Then, for any fixed $r > 1$, there is a deterministic $(1 + r)$-approximation algorithm that runs in $O(M^2 N \log^2 N \log M)$ time, and a randomized version that runs in $O(MN \log M \log N)$ expected time. For the version of the problem with preemption, there are polynomial-time approximation schemes that run in $O(MN^2 \log^2 N)$ time and $O(MN \log N \log M)$ expected time in the deterministic and randomized case respectively.

A well-known lower bound for the metric Traveling Salesman Problem (metric TSP) on N nodes is the Held-Karp bound [2], that can be formulated as the optimum of a linear program over the *subtour elimination polytope*. By using a randomized minimum-cut algorithm by Karger and Stein [3], one can obtain a randomized approximation scheme that computes the Held-Karp bound in $O(N^4 \log^6 N)$ expected time.

Open Problems

The main open problem is the further reduction of the running time for the approximate solution of the various fractional problems. One direction would be to improve the bounds for specific problems, as has been done very successfully for the multicommodity flow problem in a series of papers starting with Shahrokhi and Matula [8]. This same starting point also led to a series of results by Grigoriadis and Khachiyan developed independently to [7], starting with [1] which presents an algorithm with a number of calls smaller than the one in Theorem 1 by a factor of $\log(m\varepsilon^{-1})/\log m$. Considerable effort has been dedicated to the reduction of the dependence of the running time on the width of the problem or the reduction of the width itself (for example, see [9] for sequential and parallel algorithms for mixed packing and covering), so this can be another direction of improvement.

A problem left open by [7] is the development of approximation schemes for the RAM model, that use only *polylogarithmic in the input length* precision and work for the general case of the problems considered.

Cross References

► Minimum Makespan on Unrelated Machines

Recommended Reading

1. Grigoriadis, M.D., Khachiyan, L.G.: Fast approximation schemes for convex programs with many blocks and coupling constraints. SIAM J. Optim. **4**, 86–107 (1994)
2. Held, M., Karp, R.M.: The traveling-salesman problem and minimum cost spanning trees. Oper. Res. **18**, 1138–1162 (1970)
3. Karger, D.R., Stein, C.: An $\tilde{O}(n^2)$ algorithm for minimum cut. In: Proceeding of 25th Annual ACM Symposium on Theory of Computing (STOC), 1993, pp. 757–765
4. Leighton, F.T., Makedon, F., Plotkin, S.A., Stein, C., Tardos, É., Tragoudas, S.: Fast approximation algorithms for multicommodity flow problems. J. Comp. Syst. Sci. **50**(2), 228–243 (1995)
5. Lenstra, J.K., Shmoys, D.B., Tardos, É.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. Ser. A **24**, 259–272 (1990)
6. Plotkin, S.A., Shmoys, D.B., Tardos, É.: Fast approximation algorithms for fractional packing and covering problems. In: Proceedings of 32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1991, pp. 495–504
7. Plotkin, S.A., Shmoys, D.B., Tardos, É.: Fast approximation algorithms for fractional packing and covering problems. Math. Oper. Res. **20**(2) 257–301 (1995). Preliminary version appeared in [6]
8. Shahrokhi, F., Matula, D.W.: The maximum concurrent flow problem. J. ACM **37**, 318–334 (1990)
9. Young, N.E.: Sequential and parallel algorithms for mixed packing and covering. In: Proceedings of 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2001, pp. 538–546

Full-Text Index Construction

► Suffix Array Construction
► Suffix Tree Construction in Hierarchical Memory
► Suffix Tree Construction in RAM

Fully Dynamic All Pairs Shortest Paths

2004; Demetrescu, Italiano

Giuseppe F. Italiano
Department of Information and Computer Systems, University of Rome, Rome, Italy

Problem Definition

The problem is concerned with efficiently maintaining information about all-pairs shortest paths in a dynamically changing graph. This problem has been investigated since

the 60s [17,18,20], and plays a crucial role in many applications, including network optimization and routing, traffic information systems, databases, compilers, garbage collection, interactive verification systems, robotics, dataflow analysis, and document formatting.

A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is said to be *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only. In this entry, fully dynamic algorithms for maintaining shortest paths on general directed graphs are presented.

In the *fully dynamic All Pairs Shortest Path (APSP) problem* one wishes to maintain a directed graph $G = (V, E)$ with real-valued edge weights under an intermixed sequence of the following operations:

***Update*(x, y, w):** update the weight of edge (x, y) to the real value w; this includes as a special case both edge insertion (if the weight is set from $+\infty$ to $w < +\infty$) and edge deletion (if the weight is set to $w = +\infty$);
***Distance*(x, y):** output the shortest distance from x to y.
***Path*(x, y):** report a shortest path from x to y, if any.

More formally, the problem can be defined as follows.

Problem 1 (Fully Dynamic All-Pairs Shortest Paths) INPUT: *A weighted directed graph $G = (V, E)$, and a sequence σ of operations as defined above.*

OUTPUT: *A matrix D such entry $D[x, y]$ stores the distance from vertex x to vertex y throughout the sequence σ of operations.*

Throughout this entry, m and n denotes respectively the number of edges and vertices in G.

Demetrescu and Italiano [3] proposed a new approach to dynamic path problems based on maintaining classes of paths characterized by local properties, i. e., properties that hold for all proper subpaths, even if they may not hold for the entire paths. They showed that this approach can play a crucial role in the dynamic maintenance of shortest paths.

Key Results

Theorem 1 *The fully dynamic shoretest path problem can be solved in $O(n^2 \log^3 n)$ amortized time per update during any intermixed sequence of operations. The space required is $O(mn)$.*

Using the same approach, Thorup [22] has shown how to slightly improve the running times:

Theorem 2 *The fully dynamic shoretest path problem can be solved in $O(n^2(\log n + \log^2(m/n)))$ amortized time per update during any intermixed sequence of operations. The space required is $O(mn)$.*

Applications

Dynamic shortest paths find applications in many areas, including network optimization and routing, transportation networks, traffic information systems, databases, compilers, garbage collection, interactive verification systems, robotics, dataflow analysis, and document formatting.

Open Problems

The recent work on dynamic shortest paths has raised some new and perhaps intriguing questions. First, can one reduce the space usage for dynamic shortest paths to $O(n^2)$? Second, and perhaps more importantly, can one solve efficiently fully dynamic *single-source* reachability and shortest paths on general graphs? Finally, are there any general techniques for making increase-only algorithms fully dynamic? Similar techniques have been widely exploited in the case of fully dynamic algorithms on undirected graphs [11,12,13].

Experimental Results

A thorough empirical study of the algorithms described in this entry is carried out in [4].

Data Sets

Data sets are described in [4].

Cross References

- ► Dynamic Trees
- ► Fully Dynamic Connectivity
- ► Fully Dynamic Higher Connectivity
- ► Fully Dynamic Higher Connectivity for Planar Graphs
- ► Fully Dynamic Minimum Spanning Trees
- ► Fully Dynamic Planarity Testing
- ► Fully Dynamic Transitive Closure

Recommended Reading

1. Ausiello, G., Italiano, G.F., Marchetti-Spaccamela, A., Nanni, U.: Incremental algorithms for minimal length paths. J. Algorithm **12**(4), 615–38 (1991)
2. Demetrescu, C.: Fully Dynamic Algorithms for Path Problems on Directed Graphs. Ph. D. thesis, Department of Computer and Systems Science, University of Rome "La Sapienza", Rome (2001)
3. Demetrescu, C., Italiano, G.F.: A new approach to dynamic all pairs shortest paths. J. Assoc. Comp. Mach. **51**(6), 968–992 (2004)
4. Demetrescu, C., Italiano, G.F.: Experimental analysis of dynamic all pairs shortest path algorithms. ACM Trans. Algorithms **2**(4), 578–601 (2006)
5. Demetrescu, C., Italiano, G.F.: Trade-offs for fully dynamic reachability on dags: Breaking through the $O(n^2)$ barrier. J. Assoc. Comp. Mach. **52**(2), 147–156 (2005)
6. Demetrescu, C., Italiano, G.F.: Fully Dynamic All Pairs Shortest Paths with Real Edge Weights. J. Comp. Syst. Sci. **72**(5), 813–837 (2006)
7. Even, S., Gazit, H.: Updating distances in dynamic graphs. Method. Oper. Res. **49**, 371–387 (1985)
8. Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Semi-dynamic algorithms for maintaining single source shortest paths trees. Algorithmica **22**(3), 250–274 (1998)
9. Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Fully dynamic algorithms for maintaining shortest paths trees. J. Algorithm **34**, 351–381 (2000)
10. Henzinger, M., King, V.: Fully dynamic biconnectivity and transitive closure. In: Proc. 36th IEEE Symposium on Foundations of Computer Science (FOCS'95). IEEE Computer Society, pp. 664–672. Los Alamos (1995)
11. Henzinger, M., King, V.: Maintaining minimum spanning forests in dynamic graphs. SIAM J. Comp. **31**(2), 364–374 (2001)
12. Henzinger, M.R., King, V.: Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J. ACM **46**(4), 502–516 (1999)
13. Holm, J., de Lichtenberg, K., Thorup, M.: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J. ACM **48**, 723–760 (2001)
14. King, V.: Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In: Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS'99). IEEE Computer Society pp. 81–99. Los Alamos (1999)
15. King, V., Sagert, G.: A fully dynamic algorithm for maintaining the transitive closure. J. Comp. Syst. Sci. **65**(1), 150–167 (2002)
16. King, V., Thorup, M.: A space saving trick for directed dynamic transitive closure and shortest path algorithms. In: Proceedings of the 7th Annual International Computing and Combinatorics Conference (COCOON). LNCS, vol. 2108, pp. 268–277. Springer, Berlin (2001)
17. Loubal, P.: A network evaluation procedure. Highway Res. Rec. **205**, 96–109 (1967)
18. Murchland, J.: The effect of increasing or decreasing the length of a single arc on all shortest distances in a graph. Technical report, LBS-TNT-26, London Business School, Transport Network Theory Unit, London (1967)
19. Ramalingam, G., Reps, T.: An incremental algorithm for a generalization of the shortest path problem. J. Algorithm **21**, 267–305 (1996)
20. Rodionov, V.: The parametric problem of shortest distances. USSR Comp. Math. Math. Phys. **8**(5), 336–343 (1968)
21. Rohnert, H.: A dynamization of the all-pairs least cost problem. In: Proc. 2nd Annual Symposium on Theoretical Aspects of Computer Science, (STACS'85). LNCS, vol. 182, pp. 279–286. Springer, Berlin (1985)
22. Thorup, M.: Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In: Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT'04), pp. 384–396. Springer, Berlin (2004)
23. Thorup, M.: Worst-case update times for fully-dynamic all-pairs shortest paths. In: Proceedings of the 37th ACM Symposium on Theory of Computing (STOC 2005), ACM. New York (2005)

Fully Dynamic Connectivity

2001; Holm, de Lichtenberg, Thorup

Valerie King
Department of Computer Science, University of Victoria, Victoria, BC, Canada

Keywords and Synonyms

Incremental algorithms for graphs; Fully dynamic graph algorithm for maintaining connectivity

Problem Definition

Design a data structure for an undirected graph with a fixed set of nodes which can process queries of the form "Are nodes i and j connected?" and updates of the form "Insert edge $\{i, j\}$"; "Delete edge $\{i, j\}$." The goal is to minimize update and query times, over the worst-case sequence of queries and updates. Algorithms to solve this problem are called "fully dynamic" as opposed to "partially dynamic" since both insertions and deletions are allowed.

Key Results

Holm et al. [4] gave the first deterministic fully dynamic graph algorithm for maintaining connectivity in an undirected graph with polylogarithmic amortized time per operation, specifically, $O(\log^2 n)$ amortized cost per update operation and $O(\log n/\log\log n)$ worst-case per query, where n is the number of nodes. The basic technique is extended to maintain minimum spanning trees in $O(\log^4 n)$ amortized cost per update operation, and 2-edge connectivity and biconnectivity in $O(\log^5 n)$ amortized time per operation.

The algorithm relies on a simple novel technique for maintaining a spanning forest in a graph which enables

efficient search for a replacement edge when a tree edge is deleted. This technique ensures that each nontree edge is examined no more than $\log_2 n$ times. The algorithm relies on previously known tree data structures, such as top trees or ET-trees to store and quickly retrieve information about the spanning trees and the nontree edges incident to them.

Algorithms to achieve a query time $O(\log n/\log\log\log n)$ and expected amortized update time $O(\log n(\log\log n)^3)$ for connectivity and $O(\log^3 n \log\log n)$ expected amortized update time for 2-edge and biconnectivity were given in [6]. Lower bounds showing a continuum of tradeoffs for connectivity between query and update times in the cell probe model which match the known upper bounds were proved in [5]. Specifically, if t_u and t_q are the amortized update and query time, respectively, then $t_q \cdot \lg(t_u/t_q) = \Omega(\lg n)$ and $t_u \cdot \lg(t_q/t_u) = \Omega(\lg n)$.

A previously known, somewhat different randomized method for computing dynamic connectivity with $O(\log^3 n)$ amortized expected update time can be found in [2], improved to $O(\log^2 n)$ in [3]. A method which minimizes worst-case rather than amortized update time is given in [1]: $O(\sqrt{n})$ time per update for connectivity, as well as 2-edge connectivity and bipartiteness.

Open Problems

Can the worst-case update time be reduced to $o(n^{1/2})$, with polylogarithmic query time?

Can the lower bounds on the tradeoffs in [6] be matched for all possible query costs?

Applications

Dynamic connectivity has been used as a subroutine for several static graph algorithms, such as the maximum flow problem in a static graph [7], and for speeding up numerical studies of the Potts spin model.

URL to Code

See http://www.mpi-sb.mpg.de/LEDA/friends/dyngraph.html for software which implements the algorithm in [2] and other older methods.

Cross References

- ► Fully Dynamic All Pairs Shortest Paths
- ► Fully Dynamic Transitive Closure

Recommended Reading

1. Eppstein, D., Galil, Z., Italiano, G.F., Nissenzweig, A.:. Sparsification–a technique for speeding up dynamic graph algorithms. J. ACM **44**(5), 669–696.1 (1997)
2. Henzinger, M.R., King, V.: Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J. ACM **46**(4), 502–536 (1999) (presented at ACM STOC 1995)
3. Henzinger, M.R., Thorup, M.: Sampling to provide or to bound: With applications to fully dynamic graph algorithms. Random Struct. Algorithms **11**(4), 369–379 (1997) (presented at ICALP 1996)
4. Holm, J., De Lichtenberg, K., Thorup, M.: Poly-logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity. J. ACM **48**(4), 723–760 (2001) (presented at ACM STOC 1998)
5. Iyer, R., Karger, D., Rahul, H., Thorup, M.: An experimental study of poly-logarithmic fully-dynamic connectivity algorithms. J. Exp. Algorithmics **6**(4) (2001) (presented at ALENEX 2000)
6. Pătraşcu, M., Demaine, E.: Logarithmic Lower Bounds in the Cell-Probe Model. SIAM J. Comput. **35**(4), 932–963 (2006) (presented at ACM STOC 2004)
7. Thorup, M.: Near-optimal fully-dynamic graph connectivity. In: Proceedings of the 32th ACM Symposium on Theory of Computing pp. 343–350. ACM STOC (2000)
8. Thorup, M.: Dynamic Graph Algorithms with Applications. In: Halldórsson, M.M. (ed) 7th Scandinavian Workshop on Algorithm Theory (SWAT), Norway, 5–7 July 2000, pp. 1–9
9. Zaroliagis, C.D.: Implementations and experimental studies of dynamic graph algorithms. In: Experimental Algorithmics, Dagstuhl seminar, September 2000, Lecture Notes in Computer Science, vol. 2547. Springer (2002), Journal Article: J. Exp. Algorithmics 229–278 (2000)

Fully Dynamic Connectivity: Upper and Lower Bounds

2000; Thorup

Giuseppe F. Italiano
Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords and Synonyms

Dynamic connected components; Dynamic spanning forests

Problem Definition

The problem is concerned with efficiently maintaining information about connectivity in a dynamically changing graph. A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is said to be *fully dynamic* if it can handle both edge insertions and edge

deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

In the fully dynamic connectivity problem, one wishes to maintain an undirected graph $G = (V, E)$ under an intermixed sequence of the following operations:

***Connected*(u, v):** Return *true* if vertices u and v are in the same connected component of the graph. Return *false* otherwise.

***Insert*(x, y):** Insert a new edge between the two vertices x and y.

***Delete*(x, y):** Delete the edge between the two vertices x and y.

Key Results

In this section, a high level description of the algorithm for the fully dynamic connectivity problem in undirected graphs described in [11] is presented: the algorithm, due to Holm, de Lichtenberg and Thorup, answers connectivity queries in $O(\log n/\log\log n)$ worst-case running time while supporting edge insertions and deletions in $O(\log^2 n)$ amortized time.

The algorithm maintains a spanning forest F of the dynamically changing graph G. Edges in F are referred to as *tree edges*. Let e be a tree edge of forest F, and let T be the tree of F containing it. When e is deleted, the two trees T_1 and T_2 obtained from T after the deletion of e can be reconnected if and only if there is a non-tree edge in G with one endpoint in T_1 and the other endpoint in T_2. Such an edge is called a *replacement edge* for e. In other words, if there is a replacement edge for e, T is reconnected via this replacement edge; otherwise, the deletion of e creates a new connected component in G.

To accommodate systematic search for replacement edges, the algorithm associates to each edge e a level $\ell(e)$ and, based on edge levels, maintains a set of sub-forests of the spanning forest F: for each level i, forest F_i is the subforest induced by tree edges of level $\geq i$. Denoting by L denotes the maximum edge level, it follows that:

$$F = F_0 \supseteq F_1 \supseteq F_2 \supseteq \cdots \supseteq F_L .$$

Initially, all edges have level 0; levels are then progressively increased, but never decreased. The changes of edge levels are accomplished so as to maintain the following invariants, which obviously hold at the beginning.

Invariant (1): F is a maximum spanning forest of G if edge levels are interpreted as weights.

Invariant (2): The number of nodes in each tree of F_i is at most $n/2^i$.

Invariant (1) should be interpreted as follows. Let (u, v) be a non-tree edge of level $\ell(u, v)$ and let $u \cdots v$ be the unique path between u and v in F (such a path exists since F is a spanning forest of G). Let e be any edge in $u \cdots v$ and let $\ell(e)$ be its level. Due to (1), $\ell(e) \geq \ell(u, v)$. Since this holds for each edge in the path, and by construction $F_{\ell(u,v)}$ contains all the tree edges of level $\geq \ell(u, v)$, the entire path is contained in $F_{\ell(u,v)}$, i. e., u and v are connected in $F_{\ell(u,v)}$.

Invariant (2) implies that the maximum number of levels is $L \leq \lfloor \log_2 n \rfloor$.

Note that when a new edge is inserted, it is given level 0. Its level can be then increased at most $\lfloor \log_2 n \rfloor$ times as a consequence of edge deletions. When a tree edge $e = (v, w)$ of level $\ell(e)$ is deleted, the algorithm looks for a replacement edge at the highest possible level, if any. Due to invariant (1), such a replacement edge has level $\ell \leq \ell(e)$. Hence, a replacement subroutine `Replace`$((u, w),\ell(e))$ is called with parameters e and $\ell(e)$. The operations performed by this subroutine are now sketched.

`Replace`$((u, w), \ell)$ finds a replacement edge of the highest level $\leq \ell$, if any. If such a replacement does not exist in level ℓ, there are two cases: if $\ell > 0$, the algorithm recurses on level $\ell - 1$; otherwise, $\ell = 0$, and the deletion of (v, w) disconnects v and w in G.

During the search at level ℓ, suitably chosen tree and non-tree edges may be promoted at higher levels as follows. Let T_v and T_w be the trees of forest F_ℓ obtained after deleting (v, w) and let, w.l.o.g., T_v be smaller than T_w. Then T_v contains at most $n/2^{\ell+1}$ vertices, since $T_v \cup T_w \cup \{(v, w)\}$ was a tree at level ℓ and due to invariant (2). Thus, edges in T_v of level ℓ can be promoted at level $\ell + 1$ by maintaining the invariants. Non-tree edges incident to T_v are finally visited one by one: if an edge does connect T_v and T_w, a replacement edge has been found and the search stops, otherwise its level is increased by 1.

Trees of each forest are maintained so that the basic operations needed to implement edge insertions and deletions can be supported in $O(\log n)$ time. There are few variants of basic data structures that can accomplish this task, and one could use the Euler Tour trees (in short ET-tree), first introduced in [17], for this purpose.

In addition to inserting and deleting edges from a forest, ET-trees must also support operations such as finding the tree of a forest that contains a given vertex, computing the size of a tree, and, more importantly, finding tree edges of level ℓ in T_v and non-tree edges of level ℓ incident to T_v. This can be done by augmenting the ET-trees with

a constant amount of information per node: the interested reader is referred to [11] for details.

Using an amortization argument based on level changes, the claimed $O(\log^2 n)$ bound on the update time can be proved. Namely, inserting an edge costs $O(\log n)$, as well as increasing its level. Since this can happen $O(\log n)$ times, the total amortized insertion cost, inclusive of level increases, is $O(\log^2 n)$. With respect to edge deletions, cutting and linking $O(\log n)$ forest has a total cost $O(\log^2 n)$; moreover, there are $O(\log n)$ recursive calls to `Replace`, each of cost $O(\log n)$ plus the cost amortized over level increases. The ET-trees over $F_0 = F$ allows it to answer connectivity queries in $O(\log n)$ worst-case time. As shown in [11], this can be reduced to $O(\log n/\log\log n)$ by using a $\Theta(\log n)$-ary version of ET-trees.

Theorem 1 *A dynamic graph G with n vertices can be maintained upon insertions and deletions of edges using $O(\log^2 n)$ amortized time per update and answering connectivity queries in $O(\log n/\log\log n)$ worst-case running time.*

Later on, Thorup [18] gave another data structure which achieves slightly different time bounds:

Theorem 2 *A dynamic graph G with n vertices can be maintained upon insertions and deletions of edges using $O(\log n \cdot (\log\log n)^3)$ amortized time per update and answering connectivity queries in $O(\log n/\log\log\log n)$ time.*

The bounds given in Theorems 1 and 2 are not directly comparable, because each sacrifices the running time of one operation (either query or update) in order to improve the other.

The best known lower bound for the dynamic connectivity problem holds in the bit-probe model of computation and is due to Pătraşcu and Tarniţă [16]. The bit-probe model is an instantiation of the cell-probe model with one-bit cells. In this model, memory is organized in cells, and the algorithms may read or write a cell in constant time. The number of cell probes is taken as the measure of complexity. For formal definitions of this model, the interested reader is referred to [13].

Theorem 3 *Consider a bit-probe implementation for dynamic connectivity, in which updates take expected amortized time t_u, and queries take expected time t_q. Then, in the average case of an input distribution, $t_u = \Omega\left(\log^2 n/\log^2(t_u + t_q)\right)$. In particular*

$$\max\{t_u, t_q\} = \Omega\left(\left(\frac{\log n}{\log\log n}\right)^2\right).$$

In the bit-probe model, the best upper bound per operation is given by the algorithm of Theorem 2, namely it is $O(\log^2 n/\log\log\log n)$. Consequently, the gap between upper and lower bound appears to be limited essentially to doubly logarithmic factors only.

Applications

Dynamic graph connectivity appears as a basic subproblem of many other important problems, such as the dynamic maintenance of minimum spanning trees and dynamic edge and vertex connectivity problems. Furthermore, there are several applications of dynamic graph connectivity in other disciplines, ranging from Computational Biology, where dynamic graph connectivity proved to be useful for the dynamic maintenance of protein molecular surfaces as the molecules undergo conformational changes [6], to Image Processing, when one is interested in maintaining the connected components of a bitmap image [3].

Open Problems

The work on dynamic connectivity raises some open and perhaps intruiguing questions. The first natural open problem is whether the gap between upper and lower bounds can be closed. Note that the lower bound of Theorem 3 seems to imply that different trade-offs between queries and updates could be possible: can we design a data structure with $o(\log n)$ time per update and $O(\text{poly}(\log n))$ per query? This would be particulary interesting in applications where the total number of queries is substantially larger than the number of updates.

Finally, is it possible to design an algorithm with matching $O(\log n)$ update and query bounds for general graphs? Note that this is possible in the special case of plane graphs [5].

Experimental Results

A thorough empirical study of dynamic connectivity algorithms has been carried out in [1,12].

Data Sets

Data sets are described in [1,12].

Cross References

- ► Dynamic Trees
- ► Fully Dynamic All Pairs Shortest Paths
- ► Fully Dynamic Higher Connectivity
- ► Fully Dynamic Higher Connectivity for Planar Graphs

- ▶ Fully Dynamic Minimum Spanning Trees
- ▶ Fully Dynamic Planarity Testing
- ▶ Fully Dynamic Transitive Closure

Recommended Reading

1. Alberts, D., Cattaneo, G., Italiano, G.F.: An empirical study of dynamic graph algorithms. ACM J. Exp. Algorithmics **2** (1997)
2. Beame, P., Fich, F.E.: Optimal bounds for the predecessor problem and related problems. J. Comp. Syst. Sci. **65**(1), 38–72 (2002)
3. Eppstein, D.: Dynamic Connectivity in Digital Images. Inf. Process. Lett. **62**(3), 121–126 (1997)
4. Eppstein, D., Galil, Z., Italiano, G.F., Nissenzweig, A.: Sparsification – a technique for speeding up dynamic graph algorithms. J. Assoc. Comp. Mach. **44**(5), 669–696 (1997)
5. Eppstein, D., Italiano, G.F., Tamassia, R., Tarjan, R.E., Westbrook, J., Yung, M.: Maintenance of a minimum spanning forest in a dynamic plane graph. J. Algorithms **13**, 33–54 (1992)
6. Eyal, E., Halperin, D.: Improved Maintenance of Molecular Surfaces Using Dynamic Graph Connectivity. in: Proc. 5th International Workshop on Algorithms in Bioinformatics (WABI 2005), Mallorca, Spain, 2005, pp. 401–413
7. Frederickson, G.N.: Data structures for on-line updating of minimum spanning trees. SIAM J. Comp. **14**, 781–798 (1985)
8. Frederickson, G.N.: Ambivalent data structures for dynamic 2-edge-connectivity and *k* smallest spanning trees. In: Proc. 32nd Symp. Foundations of Computer Science, 1991, pp. 632–641
9. Henzinger, M.R., Fredman, M.L.: Lower bounds for fully dynamic connectivity problems in graphs. Algorithmica **22**(3), 351–362 (1998)
10. Henzinger, M.R., King, V.: Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J. ACM **46**(4), 502–516 (1999)
11. Holm, J., de Lichtenberg, K., Thorup, M.: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J. ACM **48**, 723–760 (2001)
12. Iyer, R., Karger, D., Rahul, H., Thorup, M.: An Experimental Study of Polylogarithmic, Fully Dynamic, Connectivity Algorithms. ACM J. Exp. Algorithmics **6** (2001)
13. Miltersen, P.B.: Cell probe complexity – a survey. In: 19th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS), Advances in Data Structures Workshop, 1999
14. Miltersen, P.B., Subramanian, S., Vitter, J.S., Tamassia, R.: Complexity models for incremental computation. In: Ausiello, G., Italiano, G.F. (eds.) Special Issue on Dynamic and On-line Algorithms. Theor. Comp. Sci. **130**(1), 203–236 (1994)
15. Pătraşcu, M., Demain, E.D.: Lower Bounds for Dynamic Connectivity. In: Proc. 36th ACM Symposium on Theory of Computing (STOC), 2004, pp. 546–553
16. Pătraşcu, M., Tarniţă, C.: On Dynamic Bit-Probe Complexity, Theoretical Computer Science, Special Issue on ICALP'05. In: Italiano, G.F., Palamidessi, C. (eds.) vol. 380, pp. 127–142 (2007) A preliminary version in Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP'05), 2005, pp. 969–981
17. Tarjan, R.E., Vishkin, U.: An efficient parallel biconnectivity algorithm. SIAM J. Comp. **14**, 862–874 (1985)
18. Thorup, M.: Near-optimal fully-dynamic graph connectivity. In: Proc. 32nd ACM Symposium on Theory of Computing (STOC), 2000, pp. 343–350

Fully Dynamic Higher Connectivity

1997; Eppstein, Galil, Italiano, Nissenzweig

GIUSEPPE F. ITALIANO
Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords and Synonyms

Fully dynamic edge connectivity; Fully dynamic vertex connectivity

Problem Definition

The problem is concerned with efficiently maintaining information about edge and vertex connectivity in a dynamically changing graph. Before defining formally the problems, a few preliminary definitions follow.

Given an undirected graph $G = (V, E)$, and an integer $k \geq 2$, a pair of vertices $\langle u, v \rangle$ is said to be *k-edge-connected* if the removal of any $(k-1)$ edges in G leaves u and v connected. It is not difficult to see that this is an equivalence relationship: the vertices of a graph G are partitioned by this relationship into equivalence classes called *k-edge-connected components*. G is said to be *k-edge-connected* if the removal of any $(k-1)$ edges leaves G connected. As a result of these definitions, G is k-edge-connected if and only if any two vertices of G are k-edge-connected. An edge set $E' \subseteq E$ is an *edge-cut for vertices x and y* if the removal of all the edges in E' disconnects G into two graphs, one containing x and the other containing y. An edge set $E' \subseteq E$ is an *edge-cut for G* if the removal of all the edges in E' disconnects G into two graphs. An edge-cut E' for G (for x and y, respectively) is *minimal* if removing any edge from E' reconnects G (for x and y, respectively). The cardinality of an edge-cut E', denoted by $|E'|$, is given by the number of edges in E'. An edge-cut E' for G (for x and y, respectively) is said to be a *minimum cardinality edge-cut* or in short a *connectivity edge-cut* if there is no other edge-cut E'' for G (for x and y respectively) such that $|E''| < |E'|$. Connectivity edge-cuts are of course minimal edge-cuts. Note that G is k-edge-connected if and only if a connectivity edge-cut for G contains at least k edges, and vertices x and y are k-edge-connected if and only if a connectivity edge-cut for x and y contains at least k edges. A connectivity edge-cut of cardinality 1 is called a *bridge*.

The following theorem due to Ford and Fulkerson, and Elias, Feinstein and Shannon (see [7]) gives another characterization of k-edge connectivity.

Theorem 1 (Ford and Fulkerson, Elias, Feinstein and Shannon) *Given a graph G and two vertices x and y in G, x and y are k-edge-connected if and only if there are at least k edge-disjoint paths between x and y.*

In a similar fashion, a vertex set $V' \subseteq V - \{x, y\}$ is said to be a *vertex-cut* for vertices x and y if the removal of all the vertices in V' disconnects x and y. $V' \subset V$ is a *vertex-cut* for vertices G if the removal of all the vertices in V' disconnects G.

The cardinality of a vertex-cut V', denoted by $|V'|$, is given by the number of vertices in V'. A vertex-cut V' for x and y is said to be a *minimum cardinality vertex-cut* or in short a *connectivity vertex-cut* if there is no other vertex-cut V'' for x and y such that $|V''| < |V'|$. Then x and y are k-vertex-connected if and only if a connectivity vertex-cut for x and y contains at least k vertices. A graph G is said to be *k-vertex-connected* if all its pairs of vertices are k-vertex-connected. A connectivity vertex-cut of cardinality 1 is called an *articulation point*, while a connectivity vertex-cut of cardinality 2 is called a *separation pair*. Note that for vertex connectivity it is no longer true that the removal of a connectivity vertex-cut splits G into two sets of vertices.

The following theorem due to Menger (see [7]) gives another characterization of k-vertex connectivity.

Theorem (Menger) 2 *Given a graph G and two vertices x and y in G, x and y are k-vertex-connected if and only if there are at least k vertex-disjoint paths between x and y.*

A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

In the *fully dynamic k-edge connectivity problem* one wishes to maintain an undirected graph $G = (V, E)$ under an intermixed sequence of the following operations:

- *k-EdgeConnected(u, v):* Return *true* if vertices u and v are in the same k-edge-connected component. Return *false* otherwise.
- *Insert(x, y):* Insert a new edge between the two vertices x and y.
- *Delete(x, y):* Delete the edge between the two vertices x and y.

In the *fully dynamic k-vertex connectivity problem* one wishes to maintain an undirected graph $G = (V, E)$ under an intermixed sequence of the following operations:

- *k-VertexConnected(u, v):* Return *true* if vertices u and v are k-vertex-connected. Return *false* otherwise.
- *Insert(x, y):* Insert a new edge between the two vertices x and y.
- *Delete(x, y):* Delete the edge between the two vertices x and y.

Key Results

To the best knowledge of the author, the most efficient fully dynamic algorithms for k-edge and k-vertex connectivity were proposed in [3,12]. Their running times are characterized by the following theorems.

Theorem 3 *The fully dynamic k-edge connectivity problem can be solved in:*

1. $O(\log^4 n)$ *time per update and* $O(\log^3 n)$ *time per query, for* $k = 2$
2. $O(n^{2/3})$ *time per update and query, for* $k = 3$
3. $O(n\alpha(n))$ *time per update and query, for* $k = 4$
4. $O(n \log n)$ *time per update and query, for* $k \geq 5$.

Theorem 4 *The fully dynamic k-vertex connectivity problem can be solved in:*

1. $O(\log^4 n)$ *time per update and* $O(\log^3 n)$ *time per query, for* $k = 2$
2. $O(n)$ *time per update and query, for* $k = 3$
3. $O(n\alpha(n))$ *time per update and query, for* $k = 4$.

Applications

Vertex and edge connectivity problems arise often in issues related to network reliability and survivability. In computer networks, the vertex connectivity of the underlying graph is related to the smallest number of nodes that might fail before disconnecting the whole network. Similarly, the edge connectivity is related to the smallest number of links that might fail before disconnecting the entire network. Analogously, if two nodes are k-vertex-connected then they can remain connected even after the failure of up to $(k - 1)$ other nodes, and if they are k-edge-connected then they can survive the failure of up to $(k - 1)$ links. It is important to investigate the dynamic versions of those problems in contexts where the networks are dynamically evolving, say, when links may go up and down because of failures and repairs.

Open Problems

The work of Eppstein et al. [3] and Holm et al. [12] raises some intriguing questions. First, while efficient dynamic algorithms for k-edge connectivity are known for general k, no efficient fully dynamic k-vertex connectivity is known for $k \geq 5$. To the best of the author's knowledge, in this case even no static algorithm is known. Second, fully dynamic 2-edge and 2-vertex connectivity can be solved in polylogarithmic time per update, while the best known update bounds for higher edge and vertex connectivity are polynomial: Can this gap be reduced, i. e., can one design polylogarithnmic algorithms for fully dynamic 3-edge and 3-vertex connectivity?

Cross References

- ▶ Dynamic Trees
- ▶ Fully Dynamic All Pairs Shortest Paths
- ▶ Fully Dynamic Connectivity
- ▶ Fully Dynamic Higher Connectivity for Planar Graphs
- ▶ Fully Dynamic Minimum Spanning Trees
- ▶ Fully Dynamic Planarity Testing
- ▶ Fully Dynamic Transitive Closure

Recommended Reading

1. Dinitz, E.A.: Maintaining the 4-edge-connected components of a graph on-line. In: Proc. 2nd Israel Symp. Theory of Computing and Systems, 1993, pp. 88–99
2. Dinitz, E.A., Karzanov A.V., Lomonosov M.V.: On the structure of the system of minimal edge cuts in a graph. In: Fridman, A.A. (ed) Studies in Discrete Optimization, pp. 290–306. Nauka, Moscow (1990). In Russian
3. Eppstein, D., Galil Z., Italiano G.F., Nissenzweig A.: Sparsification – a technique for speeding up dynamic graph algorithms. J. Assoc. Comput. Mach. **44**(5), 669–696 (1997)
4. Frederickson, G.N.: Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. SIAM J. Comput. **26**(2), 484–538 (1997)
5. Galil, Z., Italiano, G. F.: Fully dynamic algorithms for 2-edge-connectivity. SIAM J. Comput. **21**, 1047–1069 (1992)
6. Galil, Z., Italiano, G.F.: Maintaining the 3-edge-connected components of a graph on-line. SIAM J. Comput. **22**, 11–28 (1993)
7. Harary, F.: Graph Theory. Addison-Wesley, Reading (1969)
8. Henzinger, M.R.: Fully dynamic biconnectivity in graphs. Algorithmica **13**(6), 503–538 (1995)
9. Henzinger, M.R.: Improved data structures for fully dynamic biconnectivity. SIAM J. Comput. **29**(6), 1761–1815 (2000)
10. Henzinger, M., King V.: Fully dynamic biconnectivity and transitive closure. In: Proc. 36th IEEE Symposium on Foundations of Computer Science (FOCS'95), 1995, pp. 664–672
11. Henzinger, M.R., King, V.: Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J. ACM **46**(4), 502–516 (1999)
12. Holm, J., de Lichtenberg, K., Thorup, M.: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J. ACM **48**, 723–760 (2001)
13. Karzanov, A.V., Timofeev, E. A.: Efficient algorithm for finding all minimal edge cuts of a nonoriented graph. Cybernetics **22**, 156–162 (1986)
14. La Poutré, J.A.: Maintenance of triconnected components of graphs. In: Proc. 19th Int. Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science, vol. 623, pp. 354–365. Springer, Berlin (1992)
15. La Poutré, J.A.: Maintenance of 2- and 3-edge-connected components of graphs II. SIAM J. Comput. **29**(5), 1521–1549 (2000)
16. La Poutré, J.A., van Leeuwen, J., Overmars, M.H.: Maintenance of 2- and 3-connected components of graphs, part I: 2- and 3-edge-connected components. Discret. Math. **114**, 329–359 (1993)
17. La Poutré, J.A., Westbrook, J.: Dynamic two-connectivity with backtracking. In: Proc. 5th ACM-SIAM Symp. Discrete Algorithms, 1994, pp. 204–212
18. Westbrook, J., Tarjan, R.E.: Maintaining bridge-connected and biconnected components on-line. Algorithmica **7**, 433–464 (1992)

Fully Dynamic Higher Connectivity for Planar Graphs

1998; Eppstein, Galil, Italiano, Spencer

Giuseppe F. Italiano
Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords and Synonyms

Fully dynamic edge connectivity; Fully dynamic vertex connectivity

Problem Definition

In this entry, the problem of maintaining a dynamic planar graph subject to edge insertions and edge deletions that preserve planarity but that can change the embedding is considered. In particular, in this problem one is concerned with the problem of efficiently maintaining information about edge and vertex connectivity in such a dynamically changing planar graph. The algorithms to solve this problem must handle insertions that keep the graph planar without regard to any particular embedding of the graph. The interested reader is referred to the chapter "Fully Dynamic Planarity Testing" of this encyclopedia for algorithms to learn how to check efficiently whether a graph subject to edge insertions and deletions remains planar (without regard to any particular embedding).

Before defining formally the problems considered here, a few preliminary definitions follow.

Given an undirected graph $G = (V, E)$, and an integer $k \geq 2$, a pair of vertices $\langle u, v \rangle$ is said to be *k-edge-connected* if the removal of any $(k-1)$ edges in G leaves u and v connected. It is not difficult to see that this is an equivalence relationship: the vertices of a graph G are partitioned by this relationship into equivalence classes called *k-edge-connected components*. G is said to be *k-edge-connected* if the removal of any $(k-1)$ edges leaves G connected. As a result of these definitions, G is k-edge-connected if and only if any two vertices of G are k-edge-connected. An edge set $E' \subseteq E$ is an *edge-cut for vertices x and y* if the removal of all the edges in E' disconnects G into two graphs, one containing x and the other containing y. An edge set $E' \subseteq E$ is an *edge-cut for G* if the removal of all the edges in E' disconnects G into two graphs. An edge-cut E' for G (for x and y, respectively) is *minimal* if removing any edge from E' reconnects G (for x and y, respectively). The cardinality of an edge-cut E', denoted by $|E'|$, is given by the number of edges in E'. An edge-cut E' for G (for x and y, respectively) is said to be a *minimum cardinality edge-cut* or in short a *connectivity edge-cut* if there is no other edge-cut E'' for G (for x and y, respectively) such that $|E''| < |E'|$. Connectivity edge-cuts are of course minimal edge-cuts. Note that G is k-edge-connected if and only if a connectivity edge-cut for G contains at least k edges, and vertices x and y are k-edge-connected if and only if a connectivity edge-cut for x and y contains at least k edges. A connectivity edge-cut of cardinality 1 is called a *bridge*.

In a similar fashion, a vertex set $V' \subseteq V - \{x, y\}$ is said to be a *vertex-cut* for vertices x and y if the removal of all the vertices in V' disconnects x and y. $V' \subset V$ is a *vertex-cut* for vertices G if the removal of all the vertices in V' disconnects G.

The cardinality of a vertex-cut V', denoted by $|V'|$, is given by the number of vertices in V'. A vertex-cut V' for x and y is said to be a *minimum cardinality vertex-cut* or in short a *connectivity vertex-cut* if there is no other vertex-cut V'' for x and y such that $|V''| < |V'|$. Then x and y are k-vertex-connected if and only if a connectivity vertex-cut for x and y contains at least k vertices. A graph G is said to be *k-vertex-connected* if all its pairs of vertices are k-vertex-connected. A connectivity vertex-cut of cardinality 1 is called an *articulation point*, while a connectivity vertex-cut of cardinality 2 is called a *separation pair*. Note that for vertex connectivity it is no longer true that the removal of a connectivity vertex-cut splits G into two sets of vertices.

A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

In the *fully dynamic k-edge connectivity problem for a planar graph* one wishes to maintain an undirected planar graph $G = (V, E)$ under an intermixed sequence of edge insertions, edge deletions and queries about the k-edge connectivity of the underlying planar graph. Similarly, in the*fully dynamic k-vertex connectivity problem for a planar graph* one wishes to maintain an undirected planar graph $G = (V, E)$ under an intermixed sequence of edge insertions, edge deletions and queries about the k-vertex connectivity of the underlying planar graph.

Key Results

The algorithms in [2,3] solve efficiently the above problems for small values of k:

Theorem 1 *One can maintain a planar graph, subject to insertions and deletions that preserve planarity, and allow queries that test the 2-edge connectivity of the graph, or test whether two vertices belong to the same 2-edge-connected component, in $O(\log n)$ amortized time per insertion or query, and $O(\log^2 n)$ per deletion.*

Theorem 2 *One can maintain a planar graph, subject to insertions and deletions that preserve planarity, and allow testing of the 3-edge and 4-edge connectivity of the graph in $O(n^{1/2})$ time per update, or testing of whether two vertices are 3- or 4-edge-connected, in $O(n^{1/2})$ time per update or query.*

Theorem 3 *One can maintain a planar graph, subject to insertions and deletions that preserve planarity, and allow queries that test the 3-vertex connectivity of the graph, or test whether two vertices belong to the same 3-vertex-connected component, in $O(n^{1/2})$ amortized time per update or query.*

Note that these theorems improve on the bounds known for the same problems on general graphs, reported in the chapter "Fully Dynamic Higher Connectivity."

Applications

The interest reader is referred to the chapter "Fully Dynamic Higher Connectivity" for applications of dynamic edge and vertex connectivity. The case of planar graphs

is especially important, as these graphs arise frequently in applications.

Open Problems

A number of problems related to the work of Eppstein et al. [2,3] remain open. First, can the running times per operation be improved? Second, as in the case of general graphs, also for planar graphs fully dynamic 2-edge connectivity can be solved in polylogarithmic time per update, while the best known update bounds for higher edge and vertex connectivity are polynomial: Can this gap be reduced, i. e., can one design polylogarithnmic algorithms at least for fully dynamic 3-edge and 3-vertex connectivity? Third, in the special case of planar graphs can one solve fully dynamic k-vertex connectivity for general k?

Cross References

- ▶ Dynamic Trees
- ▶ Fully Dynamic All Pairs Shortest Paths
- ▶ Fully Dynamic Connectivity
- ▶ Fully Dynamic Higher Connectivity
- ▶ Fully Dynamic Minimum Spanning Trees
- ▶ Fully Dynamic Planarity Testing
- ▶ Fully Dynamic Transitive Closure

Recommended Reading

1. Galil Z., Italiano G.F., Sarnak N.: Fully dynamic planarity testing with applications. J. ACM **48**, 28–91 (1999)
2. Eppstein D., Galil Z., Italiano G.F., Spencer T.H.: Separator based sparsification I: planarity testing and minimum spanning trees. J. Comput. Syst. Sci., Special issue of STOC 93 **52**(1), 3–27 (1996)
3. Eppstein D., Galil Z., Italiano G.F., Spencer T.H.: Separator based sparsification II: edge and vertex connectivity. SIAM J. Comput. **28**, 341–381 (1999)
4. Giammarresi D., Italiano G.F.: Decremental 2- and 3-connectivity on planar graphs. Algorithmica **16**(3), 263–287 (1996)
5. Hershberger J., M.R., Suri S.: Data structures for two-edge connectivity in planar graphs. Theor. Comput. Sci. **130**(1), 139–161 (1994)

Fully Dynamic Minimum Spanning Trees

2000; Holm, de Lichtenberg, Thorup

Giuseppe F. Italiano
Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords and Synonyms

Dynamic minimum spanning forests

Problem Definition

Let $G = (V, E)$ be an undirected weighted graph. The problem considered here is concerned with maintaining efficiently information about a minimum spanning tree of G (or minimum spanning forest if G is not connected), when G is subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. One expects from the dynamic algorithm to perform update operations faster than recomputing the entire minimum spanning tree from scratch.

Throughout, an algorithm is said to be *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

Key Results

The dynamic minimum spanning forest algorithm presented in this section builds upon the dynamic connectivity algorithm described in the entry "Fully Dynamic Connectivity". In particular, a few simple changes to that algorithm are sufficient to maintain a minimum spanning forest of a weighted undirected graph upon deletions of edges [13]. A general reduction from [11] can then be applied to make the deletions-only algorithm fully dynamic.

This section starts by describing a decremental algorithm for maintaining a minimum spanning forest under deletions only. Throughout the sequence of deletions, the algorithm maintains a minimum spanning forest F of the dynamically changing graph G. The edges in F are referred to as *tree edges* and the other edges (in $G - F$) are referred to as *non-tree edges*. Let e be an edge being deleted. If e is a non-tree edge, then the minimum spanning forest does not need to change, so the interesting case is when e is a tree edge of forest F. Let T be the tree of F containing e. In this case, the deletion of e disconnects the tree T into two trees T_1 and T_2: to update the minimum spanning forest, one has to look for the minimum weight edge having one endpoint in T_1 and the other endpoint in T_2. Such an edge is called a *replacement edge* for e.

As for the dynamic connectivity algorithm, to search for replacement edges, the algorithm associates to each edge e a level $\ell(e)$ and, based on edge levels, maintains a set of sub-forests of the minimum spanning forest F: for each level i, forest F_i is the sub-forest induced by tree edges of level $\geq i$. Denoting by L the maximum edge level, it follows that:

$$F = F_0 \supseteq F_1 \supseteq F_2 \supseteq \cdots \supseteq F_L .$$

Initially, all edges have level 0; levels are then progressively increased, but never decreased. The changes of edge levels are accomplished so as to maintain the following invariants, which obviously hold at the beginning.

Invariant (1): F is a maximum spanning forest of G if edge levels are interpreted as weights.

Invariant (2): The number of nodes in each tree of F_i is at most $n/2^i$.

Invariant (3): Every cycle C has a non-tree edge of maximum weight and minimum level among all the edges in C.

Invariant (1) should be interpreted as follows. Let (u,v) be a non-tree edge of level $\ell(u,v)$ and let $u \cdots v$ be the unique path between u and v in F (such a path exists since F is a spanning forest of G). Let e be any edge in $u \cdots v$ and let $\ell(e)$ be its level. Due to (1), $\ell(e) \geq \ell(u,v)$. Since this holds for each edge in the path, and by construction $F_{\ell(u,v)}$ contains all the tree edges of level $\geq \ell(u,v)$, the entire path is contained in $F_{\ell(u,v)}$, i. e., u and v are connected in $F_{\ell(u,v)}$.

Invariant (2) implies that the maximum number of levels is $L \leq \lfloor \log_2 n \rfloor$.

Invariant (3) can be used to prove that, among all the replacement edges, the lightest edge is on the maximum level. Let e_1 and e_2 be two replacement edges with $w(e_1) < w(e_2)$, and let C_i be the cycle induced by e_i in F, $i = 1, 2$. Since F is a minimum spanning forest, e_i has maximum weight among all the edges in C_i. In particular, since by hypothesis $w(e_1) < w(e_2)$, e_2 is also the heaviest edge in cycle $C = (C_1 \cup C_2) \setminus (C_1 \cap C_2)$. Thanks to Invariant (3), e_2 has minimum level in C, proving that $\ell(e_2) \leq \ell(e_1)$. Thus, considering non-tree edges from higher to lower levels is correct.

Note that initially, an edge is is given level 0. Its level can be then increased at most $\lfloor \log_2 n \rfloor$ times as a consequence of edge deletions. When a tree edge $e = (v, w)$ of level $\ell(e)$ is deleted, the algorithm looks for a replacement edge at the highest possible level, if any. Due to invariant (1), such a replacement edge has level $\ell \leq \ell(e)$. Hence, a replacement subroutine `Replace`$((u, w), \ell(e))$ is called with parameters e and $\ell(e)$. The operations performed by this subroutine are now sketched.

`Replace`$((u, w), \ell)$ finds a replacement edge of the highest level $\leq \ell$, if any, considering edges in order of increasing weight. If such a replacement does not exist in level ℓ, there are two cases: if $\ell > 0$, the algorithm recurses on level $\ell - 1$; otherwise, $\ell = 0$, and the deletion of (v,w) disconnects v and w in G.

It is possible to show that `Replace` returns a replacement edge of minimum weight on the highest possible level, yielding the following lemma:

Lemma 1 *There exists a deletions-only minimum spanning forest algorithm that can be initialized on a graph with n vertices and m edges and supports any sequence of edge deletions in $O(m \log^2 n)$ total time.*

The description of a fully dynamic algorithm which performs updates in $O(\log^4 n)$ time now follows. The reduction used to obtain a fully dynamic algorithm is a slight generalization of the construction proposed by Henzinger and King [11] and works as follows.

Lemma 2 *Suppose there is a deletions-only minimum spanning tree algorithm that, for any k and ℓ, can be initialized on a graph with k vertices and ℓ edges and supports any sequence of $\Omega(\ell)$ deletions in total time $O(\ell \cdot t(k, \ell))$, where t is a non-decreasing function. Then there exists a fully-dynamic minimum spanning tree algorithm for a graph with n nodes starting with no edges, that, for m edges, supports updates in time*

$$O\left(\log^3 n + \sum_{i=1}^{3+\log_2 m} \sum_{j=1}^{i} t\left(min\{n, 2^j\}, 2^j\right)\right).$$

The interested reader is referred to references [11] and [13] for the description of the construction that proves Lemma 2. From Lemma 1 one gets $t(k, \ell) = O(\log^2 k)$. Hence, combining Lemmas 1 and 2, the claimed result follows:

Theorem 3 *There exists a fully-dynamic minimum spanning forest algorithm that, for a graph with n vertices, starting with no edges, maintains a minimum spanning forest in $O(\log^4 n)$ amortized time per edge insertion or deletion.*

There is a lower bound of $\Omega(\log n)$ for dynamic minimum spanning tree, given by *Eppstein et al.* [6], which uses the following argument. Let A be an algorithm for maintaining a minimum spanning tree of an arbitrary (multi)graph G. Let A be such that change weight(e, Δ) returns the edge f that replace e in the minimum spanning tree, if e is replaced. Clearly, any dynamic spanning tree algorithm can be modified to return f. One can use algorithm A to sort n positive numbers $x_1, x_2, \ldots, x_n$, as follows. Construct a multigraph G consisting of two nodes connected by $(n + 1)$ edges $e_0, e_1, \ldots, e_n$, such that edge e_0 has weight 0 and edge e_i has weight x_i. The initial spanning tree is e_0. Increase the weight of e_0 to $+\infty$. Whichever edge replaces e_0, say e_i, is the edge of minimum weight. Now increase the weight of e_i to $+\infty$: the replacement of e_i gives the second smallest weight. Continuing in this fashion gives

the numbers sorted in increasing order. A similar argument applies when only edge decreases are allowed. Since Paul and Simon [14] have shown that any sorting algorithm needs $\Omega(n \log n)$ time to sort n numbers on a unit-cost random access machine whose repertoire of operations include additions, subtractions, multiplications and comparisons with 0, but not divisions or bit-wise Boolean operations, the following theorem follows.

Theorem 4 *Any unit-cost random access algorithm that performs additions, subtractions, multiplications and comparisons with* 0*, but not divisions or bit-wise Boolean operations, requires* $\Omega(\log n)$ *amortized time per operation to maintain a minimum spanning tree dynamically.*

Applications

Minimum spanning trees have applications in many areas, including network design, VLSI, and geometric optimization, and the problem of maintaining minimum spanning trees dynamically arises in such applications.

Algorithms for maintaining a minimum spanning forest of a graph can be used also for maintaining information about the connected components of a graph. There are also other applications of dynamic minimum spanning trees algorithms, which include finding the k smallest spanning trees [3,4,5,8,9], sampling spanning trees [7] and dynamic matroid intersection problems [10]. Note that the first two problems are not necessarily dynamic: however, efficient solutions for these problems need dynamic data structures.

Open Problems

The first natural open question is to ask whether the gap between upper and lower bounds for the dynamic minimum spanning tree problem can be closed. Note that this is possible in the special case of plane graphs [6].

Second, the techniques for dynamic minimum spanning trees can be extended to dynamic 2-edge and 2-vertex connectivity, which indeed can be solved in polylogarithmic time per update. Can one extend the same technique also to higher forms of connectivity? This is particularly important, since the best known update bounds for higher edge and vertex connectivity are polynomial, and it would be useful to design polylogarithnmic algorithms at least for fully dynamic 3-edge and 3-vertex connectivity.

Experimental Results

A thorough empirical study on the performance evaluation of dynamic minimum spanning trees algorithms has been carried out in [1,2].

Data Sets

Data sets are described in [1,2].

Cross References

- ▶ Dynamic Trees
- ▶ Fully Dynamic All Pairs Shortest Paths
- ▶ Fully Dynamic Connectivity
- ▶ Fully Dynamic Higher Connectivity
- ▶ Fully Dynamic Higher Connectivity for Planar Graphs
- ▶ Fully Dynamic Planarity Testing
- ▶ Fully Dynamic Transitive Closure

Recommended Reading

1. Alberts, D., Cattaneo, G., Italiano, G.F.: An empirical study of dynamic graph algorithms. ACM. J. Exp. Algorithm **2**, (1997)
2. Cattaneo, G., Faruolo, P., Ferraro Petrillo, U., Italiano, G.F.: Maintaining Dynamic Minimum Spanning Trees: An Experimental Study. In: Proceeding 4th Workshop on Algorithm Engineering and Experiments (ALENEX 02), 6–8 Jan 2002. pp. 111–125
3. Eppstein, D.: Finding the k smallest spanning trees. BIT. **32**, 237–248 (1992)
4. Eppstein, D.: Tree-weighted neighbors and geometric k smallest spanning trees. Int. J. Comput. Geom. Appl. **4**, 229–238 (1994)
5. Eppstein, D., Galil, Z., Italiano, G.F., Nissenzweig, A.: Sparsification – a technique for speeding up dynamic graph algorithms. J. Assoc. Comput. Mach. **44**(5), 669–696 (1997)
6. Eppstein, D., Italiano, G.F., Tamassia, R., Tarjan, R.E., Westbrook, J., Yung, M.: Maintenance of a minimum spanning forest in a dynamic plane graph. J. Algorithms **13**, 33–54 (1992)
7. Feder, T., Mihail, M.: Balanced matroids. In: Proceeding 24th ACM Symp. Theory of Computing, pp 26–38, Victoria, British Columbia, Canada, May 04–06 1992
8. Frederickson, G.N.: Data structures for on-line updating of minimum spanning trees. SIAM. J. Comput. **14**, 781–798 (1985)
9. Frederickson, G.N.: Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. In: Proceeding 32nd Symp. Foundations of Computer Science, pp 632–641, San Juan, Puerto Rico, October 01–04 1991
10. Frederickson, G.N., Srinivas, M.A.: Algorithms and data structures for an expanded family of matroid intersection problems. SIAM. J. Comput. **18**, 112–138 (1989)
11. Henzinger, M.R., King, V.: Maintaining minimum spanning forests in dynamic graphs. SIAM. J. Comput. **31**(2), 364–374 (2001)
12. Henzinger, M.R., King, V.: Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J. ACM **46**(4), 502–516 (1999)
13. Holm, J., de Lichtenberg, K., Thorup, M.: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J. ACM **48**, 723–760 (2001)
14. Paul, J., Simon, W.: Decision trees and random access machines. In: Symposium über Logik und Algorithmik. (1980) See also Mehlhorn, K.: Sorting and Searching, pp. 85–97. Springer, Berlin (1984)

15. Tarjan, R.E., Vishkin, U.: An efficient parallel biconnectivity algorithm. SIAM. J. Comput. **14**, 862–874 (1985)

Fully Dynamic Planarity Testing

1999; Galil, Italiano, Sarnak

GIUSEPPE F. ITALIANO
Department of Information and Computer Systems, University of Rome, Rome, Italy

Problem Definition

In this entry, the problem of maintaining a dynamic planar graph subject to edge insertions and edge deletions that preserve planarity but that can change the embedding is considered. Before formally defining the problem, few preliminary definitions follow.

A graph is *planar* if it can be embedded in the plane so that no two edges intersect. In a dynamic framework, a planar graph that is committed to an embedding is called *plane*, and the general term *planar* is used only when changes in the embedding are allowed. An edge insertion that preserves the embedding is called *embedding-preserving*, whereas it is called *planarity-preserving* if it keeps the graph planar, even though its embedding can change; finally, an edge insertion is called *arbitrary* if it is not known to preserve planarity. Extensive work on dynamic graph algorithms has used ad hoc techniques to solve a number of problems such as minimum spanning forests, 2-edge-connectivity and planarity testing for plane graphs (with embedding-preserving insertions) [5,6,7,9,10,11,12]: this entry is concerned with more general planarity-preserving updates.

The work of Galil et al. [8] and of Eppstein et al. [3] provides a general technique for dynamic planar graph problems, including those mentioned above: in all these problems, one can deal with either arbitrary or planarity-preserving insertions and therefore allow changes of the embedding.

The *fully dynamic planarity testing problem* can be defined as follows. One wishes to maintain a (not necessarily planar) graph subject to *arbitrary* edge insertions and deletions, and allow queries that test whether the graph is currently planar, or whether a potential new edge would violate planarity.

Key Results

Eppstein et al. [3] provided a way to apply the sparsification technique [2] to families of graphs that are already sparse, such as planar graphs.

The new ideas behind this technique are the following. The notion of a certificate can be expanded to a definition for graphs in which a subset of the vertices are denoted as *interesting*; these *compressed certificates* may reduce the size of the graph by removing uninteresting vertices. Using this notion, one can define a type of sparsification based on *separators*, small sets of vertices the removal of which splits the graph into roughly equal size components. Recursively finding separators in these components gives a *separator tree* which can also be used as a *sparsification tree*; the interesting vertices in each certificate will be those vertices used in separators at higher levels of the tree. The notion of a *balanced separator tree*, which also partitions the interesting vertices evenly in the tree, is introduced: such a tree can be computed in linear time, and can be maintained dynamically. Using this technique, the following results can be achieved.

Theorem 1 *One can maintain a planar graph, subject to insertions and deletions that preserve planarity, and allow queries that test whether a new edge would violate planarity, in amortized time $O(n^{1/2})$ per update or query.*

This result can be improved, in order to allow arbitrary insertions or deletions, even if they might let the graph become nonplanar, using the following approach. The data structure above can be used to maintain a planar subgraph of the given graph. Whenever one attempts to insert a new edge, and the resulting graph would be nonplanar, the algorithm does not actually perform the insertion, but instead adds the edge to a list of *nonplanar edges*. Whenever a query is performed, and the list of nonplanar edges is nonempty, the algorithm attempts once more to add those edges one at a time to the planar subgraph. The time for each successful addition can be charged to the insertion operation that put that edge in the list of nonplanar edges. As soon as the algorithm finds some edge in the list that can not be added, it stops trying to add the other edges in the list. The time for this failed insertion can be charged to the query the algorithm is currently performing. In this way the list of nonplanar edges will be empty if and only if the graph is planar, and the algorithm can test planarity even for updates in nonplanar graphs.

Theorem 2 *One can maintain a graph, subject to arbitrary insertions and deletions, and allow queries that test whether the graph is presently planar or whether a new edge would violate planarity, in amortized time $O(n^{1/2})$ per update or query.*

Applications

Planar graphs are perhaps one of the most important interesting subclasses of graphs which combine beautiful structural results with relevance in applications. In particular, planarity testing is a basic problem, which appears naturally in many applications, such as VLSI layout, graphics, and computer aided design. In all these applications, there seems to be a need for dealing with dynamic updates.

Open Problems

The $O(n^{1/2})$ bound for planarity testing is amortized. Can we improve this bound or make it worst-case?

Finally, the complexity of the algorithms presented here, and the large constant factors involved in some of the asymptotic time bounds, make some of the results unsuitable for practical applications. Can one simplify the methods while retaining similar theoretical bounds?

Cross References

- ▶ Dynamic Trees
- ▶ Fully Dynamic All Pairs Shortest Paths
- ▶ Fully Dynamic Connectivity
- ▶ Fully Dynamic Higher Connectivity
- ▶ Fully Dynamic Higher Connectivity for Planar Graphs
- ▶ Fully Dynamic Minimum Spanning Trees
- ▶ Fully Dynamic Transitive Closure

Recommended Reading

1. Cimikowski, R.: Branch-and-bound techniques for the maximum planar subgraph problem. Int. J. Computer Math. **53**, 135–147 (1994)
2. Eppstein, D., Galil, Z., Italiano, G.F., Nissenzweig, A.: Sparsification – a technique for speeding up dynamic graph algorithms. J. Assoc. Comput. Mach. **44**(5), 669–696 (1997)
3. Eppstein, D., Galil, Z., Italiano, G.F., Spencer, T.H.: Separator based sparsification I: planarity testing and minimum spanning trees. J. Comput. Syst. Sci. Special issue of STOC 93 **52**(1), 3–27 (1996)
4. Eppstein, D., Galil, Z., Italiano, G.F., Spencer, T.H.: Separator based sparsification II: edge and vertex connectivity. SIAM J. Comput. **28**, 341–381 (1999)
5. Eppstein, D., Italiano, G.F., Tamassia, R., Tarjan, R.E., Westbrook, J., Yung, M.: Maintenance of a minimum spanning forest in a dynamic plane graph. J. Algorithms **13**, 33–54 (1992)
6. Frederickson, G.N.: Data structures for on-line updating of minimum spanning trees, with applications. SIAM J. Comput. **14**, 781–798 (1985)
7. Frederickson, G.N.: Ambivalent data structures for dynamic 2-edge-connectivity and *k* smallest spanning trees. SIAM J. Comput. **26**(2), 484–538 (1997)
8. Galil, Z., Italiano, G.F., Sarnak, N.: Fully dynamic planarity testing with applications. J. ACM **48**, 28–91 (1999)
9. Giammarresi, D., Italiano, G.F.: Decremental 2- and 3-connectivity on planar graphs. Algorithmica **16**(3):263–287 (1996)
10. Hershberger, J., Suri, M.R., Suri, S.: Data structures for two-edge connectivity in planar graphs. Theor. Comput. Sci. **130**(1), 139–161 (1994)
11. Italiano, G.F., La Poutré, J.A., Rauch, M.: Fully dynamic planarity testing in planar embedded graphs. 1st Annual European Symposium on Algorithms, Bad Honnef, Germany, 30 September–2 October 1993
12. Tamassia, R.: A dynamic data structure for planar graph embedding. 15th Int. Colloq. Automata, Languages, and Programming. LNCS, vol. 317, pp. 576–590. Springer, Berlin (1988)

Fully Dynamic Transitive Closure

1999; King

Valerie King
Department of Computer Science Department,
University of Victoria,
Victoria, BC, Canada

Keywords and Synonyms

Incremental algorithms for digraphs; Fully dynamic graph algorithm for maintaining transitive closure; All-pairs dynamic reachability

Problem Definition

Design a data structure for a directed graph with a fixed set of node which can process queries of the form "Is there a path from i to j ?" and updates of the form: "Insert edge (i, j)"; "Delete edge (i, j)". The goal is to minimize update and query times, over the worst case sequence of queries and updates. Algorithms to solve this problem are called "fully dynamic" as opposed to "partially dynamic" since both insertions and deletions are allowed.

Key Results

This work [4] gives the first deterministic fully dynamic graph algorithm for maintaining the transitive closure in a directed graph. It uses $O(n^2 \log n)$ amortized time per update and $O(1)$ worst case query time where n is number of nodes in the graph. The basic technique is extended to give fully dynamic algorithms for approximate and exact all-pairs shortest paths problems.

The basic building block of these algorithms is a method of maintaining all-pairs shortest paths with in-

sertions and deletions for distances up to d. For each vertex v, a single-source shortest path tree of depth d which reach v ("In_v") and another tree of vertices which are reached by v ("Out_v") are maintained during any sequence of deletions. Each insert of a set of edges incident to v results in the rebuilding of In_v and Out_vI. For each pair of vertices x, y and each length, a count is kept of the number of v such that there is a path from x in In_v to y in Out_v of that length.

To maintain transitive closure, lg n levels of these trees are maintained for trees of depth 2, where the edges used to construct a forest on one level depend on the paths in the forest of the previous level.

Space required was reduced from $O(n^3)$ to $O(n^2)$ in [6]. A log n factor was shaved off [7,10]. Other tradeoffs between update and query time are given in [1,7,8,9,10]. A deletions only randomized transitive closure algorithm running in $O(mn)$ time overall is given by [8] where m is the initial number of edges in the graph. A simple monte carlo transitive closure algorithm for acyclic graphs is presented in [5]. Dynamic single source reachability in a digraph is presented in [8,9]. All-pairs shortest paths can be maintained with nearly the same update time [2].

Applications

None

Open Problems

Can reachability from a single source in a directed graph be maintained in $o(mn)$ time over a worst case sequence of m deletions?

Can strongly connected components be maintained in $o(mn)$ time over a worst case sequence of m deletions?

Experimental Results

Experimental results on older techniques can be found in [3].

Cross References

- ▶ All Pairs Shortest Paths in Sparse Graphs
- ▶ All Pairs Shortest Paths via Matrix Multiplication
- ▶ Fully Dynamic All Pairs Shortest Paths
- ▶ Fully Dynamic Connectivity

Recommended Reading

1. Demestrescu, C., Italiano, G.F.: Trade-offs for fully dynamic transitive closure on DAG's: breaking through the $O(n^2)$ barrier, (presented in FOCS 2000). J. ACM **52**(2), 147–156 (2005)
2. Demestrescu, C., Italiano, G.F.: A new approach to dynamic all pairs shortest paths, (presented in STOC 2003). J. ACM **51**(6), 968–992 (2004)
3. Frigioni, D., Miller, T., Nanni, U., Zaroliagis, C.D.: An experimental study of dynamic algorithms for transitive closure. ACM J Exp. Algorithms **6**(9) (2001)
4. King, V.: Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In: Proceedings of the 40th Annual IEEE Symposium on Foundation of Computer Science. ComilEEE FOCS pp. 81–91. IEEE Computer Society, New York (1999)
5. King, V., Sagert, G.: A fully dynamic algorithm for maintaining the transitive closure, (presented in FOCS 1999). JCCS **65**(1), 150–167 (2002)
6. King, V., Thorup, M.: A space saving trick for dynamic transitive closure and shortest path algorithms. In: Proceedings of the 7th Annual International Conference of Computing and Cominatorics, vol. 2108/2001, pp. 269–277. Lect. Notes Comp. Sci. COCOON Springer, Heidelberg (2001)
7. Roditty, L.: A faster and simpler fully dynamic transitive closure. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms. ACM IEEE SODA, pp. 404–412. ACM, Baltimore (2003)
8. Roditty, L., Zwick, U.: Improved dynamic reachability algorithms for directed graphs. In: Proceedings of the 43rd Annual Symposium on Foundation of Computer Science. IEEE FOCS, pp. 679–688 IEEE Computer Society, Vancouver, Canada (2002)
9. Roditty, L., Zwick, U.: A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In: Proceedings of the 36th ACM Symposium on Theory of Computing. ACM STOC, pp. 184–191 ACM, Chicago (2004)
10. Sankowski, S.: Dynamic transitive closure via dynamic matrix inverse. In: Proceedings of the 45th Annual Symposium on Foundations of Computer Science. IEEE FOCS, 509–517, IEEE Computer Society, Rome, Italy (2004)