C

Cache-Oblivious B-Tree

2005; Bender, Demaine, Farach-Colton

Rolf Fagerberg
Department of Mathematics and Computer Science,
University of Southern Denmark,
Odense, Denmark

Keywords and Synonyms

Cache-oblivious search tree; Cache-oblivious dictionary

Problem Definition

Computers contain a hierarchy of memory levels, with vastly differing access times. Hence, the time for a memory access depends strongly on what is the innermost level containing the data accessed. In analysis of algorithms, the standard RAM (or von Neumann) model cannot capture this effect, and external memory models were introduced to better model the situation. The most widely used of these models is the two-level I/O-model [1], also called the External Memory model or the Disk Access model. The I/O-model approximates the memory hierarchy by modeling two levels, with the inner level having size M, the outer level having infinite size, and transfers between the levels taking place in blocks of B consecutive elements. The cost of an algorithm is the number of memory transfers it makes.

The cache-oblivious model, introduced by Frigo et al. [18], elegantly generalizes the I/O-model to a multi-level memory model by a simple measure: the algorithm is not allowed to know the value of B and M. More precisely, a cache-oblivious algorithm is an algorithm formulated in the RAM model, but analyzed in the I/O-model, with an analysis valid for *any* value of B and M. Cache replacement is assumed to take place automatically by an optimal off-line cache replacement strategy. Since the analysis holds for any B and M, it holds for all levels simultaneously.

The subject here is that of efficient cache-oblivious data structures for the ordered dictionary problem, i. e., the problem of storing elements with keys from an ordered universe while supporting searches, insertions, deletions, and range searches. In full generality, searches are predecessor searches, returning the element with the largest key smaller than or equal to the key searched for.

Key Results

The first cache-oblivious dictionary was given by Prokop [21], who showed how to lay out a static binary tree in memory such that searches take $O(\log_B n)$ memory transfers. This layout, often called the *van Emde Boas layout* because it is reminiscent of the classic van Emde Boas data structure, also ensures that range searches take $O(\log_B n + k/B)$ memory transfers [2], where k is the size of the output. Both bounds are optimal for comparison-based searching.

The first dynamic, cache-oblivious dictionary was given by Bender et al. [10]. Making use of a variant of the van Emde Boas layout, a density maintenance algorithm of the type invented by Itai et al. [19], and weight-balanced B-trees [5], they arrived at the following results:

Theorem 1 ([10]) *There is a cache-oblivious dictionary structure supporting searches in $O(\log_B n)$ memory transfers, and insertions and deletions in amortized $O(\log_B n)$ memory transfers.*

Theorem 2 ([10]) *There is a cache-oblivious dictionary structure supporting searches in $O(\log_B n)$ memory transfers, insertions and deletions in amortized $O(\log_B n + (\log^2 n)/B)$ memory transfers, and range searches in $O(\log_B n + k/B)$ memory transfers, where k is the size of the output.*

Later, Bender et al. [7] developed a cache-oblivious structure for maintaining linked lists which supports insertion and deletion of elements in $O(1)$ memory trans-

fers and scanning of k consecutive elements in amortized $O(k/B)$ memory transfers. Combining this structure with the structure of the first theorem above, the following result can be achieved.

Theorem 3 ([7,10]) *There is a cache-oblivious dictionary structure supporting searches in $O(\log_B n)$ memory transfers, insertions and deletions in amortized $O(\log_B n)$ memory transfers, and range searches in amortized $O(\log_B n + k/B)$ memory transfers, where k is the size of the output.*

A long list of extensions of these basic cache-oblivious dictionary results has been given. We now survey these.

Bender et al. [11] and Brodal et al. [16] gave very similar proposals for reproducing the result of Theorem 2, but with significantly simpler structures (avoiding the use of weight-balanced B-trees). On the basis of exponential trees, Bender et al.[8] gave a proposal with $O(\log_B n)$ worst-case queries and updates. They also gave a solution with partial persistence, where searches (in all versions of the structure) and updates (in the latest version of the structure) require amortized $O(\log_B(m + n))$ memory transfers, where m is the number of versions and n is the number of elements in the version operated on. Bender et al. [14] extended the cache-oblivious model to a concurrent setting, and gave three proposals for cache-oblivious B-trees in this setting. Bender et al. [12] gave cache-oblivious dictionary structures exploring trade-offs between faster insertion costs and slower search cost. Franceschini and Grossi [17] showed how to achieve $O(\log_B n)$ worst-case queries and updates while using $O(1)$ space besides the space for the n elements stored. Extensions to dictionaries on other data types such as strings [13,15] and geometric data [3,4,6] have been given.

It has been shown [9] that the best-possible multiplicative constant in the $\Theta(\log_B n)$ search bound for comparison-based searching is different in the I/O-model and in the cache-oblivious model.

Applications

Dictionaries solve a fundamental data structuring problem which is part of solutions for a very high number of computational problems. Dictionaries for external memory are useful in settings where memory accesses are dominating the running time, and cache-oblivious dictionaries in particular stand out by their ability to optimize the access to all levels of an unknown memory hierarchy. This is an asset e. g. when developing programs to be run on diverse or unknown architectures (such as software libraries or programs for heterogeneous distributed computing like grid computing and projects such as SETI@home). Even on a single, known architecture, the memory parameters available to a computational process may be non-constant if several processes compete for the same memory resources. Since cache-oblivious algorithms are optimized for all parameter values, they have the potential to adapt more gracefully to these changes, and also to varying input sizes forcing different memory levels to be in use.

Open Problems

For the one-dimensional ordered dictionary problem discussed here, one notable open problem is to find a data structure achieving worst case versions of all the bounds in Theorem 3.

Experimental Results

Cache-oblivious dictionaries have been evaluated empirically in [11,13,16,20,22]. The overall conclusion of these investigations is that cache-oblivious methods easily can outperform RAM algorithms, although sometimes not as much as algorithms tuned to the specific memory hierarchy and problem size in question. On the other hand, cache-oblivious algorithms seem to perform well on all levels of the memory hierarchy, and to be more robust to changing problem sizes.

Cross References

- ► B-trees
- ► Cache-Oblivious Model
- ► Cache-Oblivious Sorting
- ► I/O-model

Recommended Reading

1. Aggarwal, A., Vitter, J.S.: The Input/Output complexity of sorting and related problems. Commun. ACM **31**(9), 1116–1127 (1988)
2. Arge, L., Brodal, G.S., Fagerberg, R.: Cache-oblivious data structures. In: Mehta, D., Sahni, S. (eds.) Handbook on Data Structures and Applications. CRC Press, Boca Raton (2005)
3. Arge, L., Brodal, G.S., Fagerberg, R., Laustsen, M.: Cache-oblivious planar orthogonal range searching and counting. In: Proc. 21st ACM Symposium on Computational Geometry, pp. 160–169. ACM, New York (2005)
4. Arge, L., de Berg, M., Haverkort, H.J.: Cache-oblivious R-trees. In: Proc. 21st ACM Symposium on Computational Geometry, pp. 170–179. ACM, New York (2005)
5. Arge, L., Vitter, J.S.: Optimal external memory interval management. SIAM J. Comput. **32**(6), 1488–1508 (2003)
6. Arge, L., Zeh, N.: Simple and semi-dynamic structures for cache-oblivious planar orthogonal range searching. In: Proc. 22nd ACM Symposium on Computational Geometry, pp. 158–166. ACM, New York (2006)

7. Bender, M., Cole, R., Demaine, E., Farach-Colton, M.: Scanning and traversing: Maintaining data for traversals in a memory hierarchy. In: Proc. 10th Annual European Symposium on Algorithms. LNCS, vol. 2461, pp. 139–151. Springer, Berlin (2002)
8. Bender, M., Cole, R., Raman, R.: Exponential structures for cache-oblivious algorithms. In: Proc. 29th International Colloquium on Automata, Languages, and Programming. LNCS, vol. 2380, pp. 195–207. Springer, Berlin (2002)
9. Bender, M.A., Brodal, G.S., Fagerberg, R., Ge, D., He, S., Hu, H., Iacono, J., Lopez-Ortiz, A.: The cost of cache-oblivious searching. In: Proc. 44th Annual IEEE Symposium on Foundations of Computer Science, pp. 271–282. IEEE Computer Society Press, Los Alamitos (2003)
10. Bender, M.A., Demaine, E.D., Farach-Colton, M.: Cache-oblivious B-trees. SIAM J. Comput. **35**(2), 341–358 (2005). Conference version appeared at FOCS (2000)
11. Bender, M.A., Duan, Z., Iacono, J., Wu, J.: A locality-preserving cache-oblivious dynamic dictionary. J. Algorithms **53**(2), 115–136 (2004). Conference version appeared at SODA (2002)
12. Bender, M.A., Farach-Colton, M., Fineman, J.T., Fogel, Y.R., Kuszmaul, B.C., Nelson, J.: Cache-oblivious streaming B-trees. In: Proc. 19th Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 81–92. ACM, New York (2007)
13. Bender, M.A., Farach-Colton, M., Kuszmaul, B.C.: Cache-oblivious string B-trees. In: Proc. 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 233–242. ACM, New York (2006)
14. Bender, M.A., Fineman, J.T., Gilbert, S., Kuszmaul, B.C.: Concurrent cache-oblivious B-trees. In: Proc. 17th Annual ACM Symposium on Parallel Algorithms, pp. 228–237. ACM, New York (2005)
15. Brodal, G.S., Fagerberg, R.: Cache-oblivious string dictionaries. In: SODA: ACM-SIAM Symposium on Discrete Algorithms, pp. 581–590. ACM Press, New York (2006)
16. Brodal, G.S., Fagerberg, R., Jacob, R.: Cache-oblivious search trees via binary trees of small height. In: Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 39–48 ACM, New York (2002)
17. Franceschini, G., Grossi, R.: Optimal worst-case operations for implicit cache-oblivious search trees. In: Proc. Algorithms and Data Structures, 8th International Workshop, WADS. LNCS, vol. 2748, pp. 114–126. Springer, Berlin (2003)
18. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: 40th Annual IEEE Symposium on Foundations of Computer Science, pp. 285–298. IEEE Computer Society Press, Los Alamitos (1999)
19. Itai, A., Konheim, A.G., Rodeh, M.: A sparse table implementation of priority queues. In: Automata, Languages and Programming, 8th Colloquium. LNCS, vol. 115, pp. 417–431. Springer, Berlin (1981)
20. Ladner, R.E., Fortna, R., B.-Nguyen, H.: A comparison of cache aware and cache oblivious static search trees using program instrumentation. In: Experimental Algorithmics. LNCS, vol. 2547, pp. 78–92. Springer, Berlin (2000)
21. Prokop, H.: Cache-oblivious algorithms. Master's thesis, Massachusetts Institute of Technology (1999)
22. Rahman, N., Cole, R., Raman, R.: Optimised predecessor data structures for internal memory. In: Proc. Algorithm Engineering, 5th International Workshop, WAE. LNCS, vol. 2141, pp. 67–78. Springer, Berlin (2001)

Cache-Oblivious Model

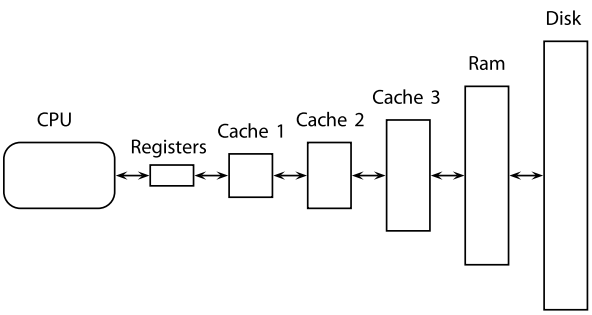
1999; Frigo, Leiserson, Prokop, Ramachandran

Rolf Fagerberg
Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

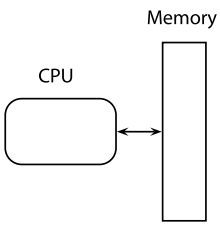
Model Definition

The memory system of contemporary computers consists of a hierarchy of memory levels, with each level acting as a cache for the next; a typical hierarchy may consist of registers, level 1 cache, level 2 cache, level 3 cache, main memory, and disk (Fig. 1). One characteristic of the hierarchy is that the memory levels get larger and slower the further they get from the processor, with the access time increasing most dramatically between RAM memory and disk. Another characteristic is that data is moved between levels in blocks.

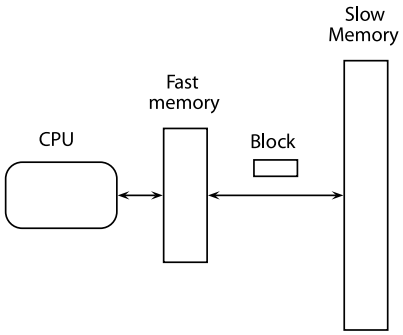
As a consequence of the differences in access time between the levels, the cost of a memory access depends highly on what is the current lowest memory level holding the element accessed. Hence, the memory access pattern of an algorithm has a major influence on its practical running time. Unfortunately, the RAM model (Fig. 2) traditionally used to design and analyze algorithms is not



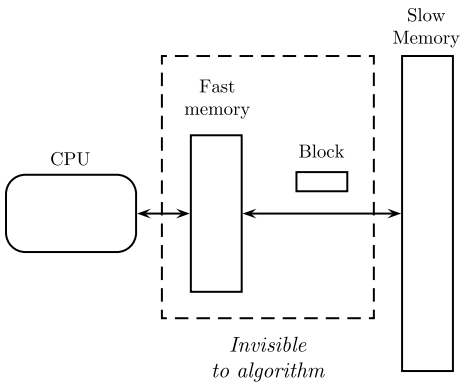
Cache-Oblivious Model, Figure 1
The memory hierarchy



Cache-Oblivious Model, Figure 2
The RAM model



Cache-Oblivious Model, Figure 3
The I/O-model



Cache-Oblivious Model, Figure 4
The cache-oblivious model

capable of capturing this, as it assumes that all memory accesses take equal time.

To better account for the effects of the memory hierarchy, a number of computational models have been proposed. The simplest and most successful is the two-level I/O-model introduced by Aggarwal and Vitter [2] (Fig. 3). In this model a two-level memory hierarchy is assumed, consisting of a fast memory of size M and a slower memory of infinite size, with data transferred between the levels in blocks of B consecutive elements. Computation can only be performed on data in the fast memory, and algorithms are assumed to have complete control over transfers of blocks between the two levels. Such a block transfer is denoted a *memory transfer*. The complexity measure is the number of memory transfers performed. The strength of the I/O-model is that it captures part of the memory hierarchy, while being sufficiently simple to make design and analysis of algorithms feasible. Over the last two decades, a large body of results for the I/O-model has been produced, covering most areas of algorithmics. For an overview, see the surveys [3,24,26,27].

More elaborate models of multi-level memory have been proposed (see e. g. [26] for an overview) but these models have been less successful than the I/O-model, mainly because of their complexity which makes analysis of algorithms harder. All these models, including the I/O-model, assume that the characteristics of the memory hierarchy (the level and block sizes) are known.

In 1999 the *cache-oblivious model* (Fig. 4) was introduced by Frigo et al. [22]. A cache-oblivious algorithm is an algorithm formulated in the RAM model but analyzed in the I/O-model, with the analysis required to hold for *any* block size B and memory size M. Memory transfers are assumed to take place automatically by an optimal off-line cache replacement strategy.

The crux of the cache-oblivious model is that because the I/O-model analysis holds for any block and memory size, it holds for all levels of a multi-level memory hierarchy (see [22,25] for detailed versions of this statement). Put differently, by optimizing an algorithm to one unknown level of the memory hierarchy, it is optimized to all levels simultaneously. Thus, the cache-oblivious model elegantly generalizes the I/O-model to a multi-level memory model by one simple measure: the algorithm is not allowed to know the value of B and M. The challenge, of course, is to develop algorithms having good memory transfer analyzes under these conditions.

Besides capturing the entire memory hierarchy in a conceptually simple way, the cache-oblivious model has other benefits: Algorithms developed in the model do not rely on knowing the parameters of the memory hierarchy, which is an asset when developing programs to be run on diverse or unknown architectures (e. g. software libraries or programs for heterogeneous distributed computing such as grid computing and projects like SETI@home). Even on a single, known architecture, the memory parameters available to a computational process may be non-constant if several processes compete for the same memory resources. Since cache-oblivious algorithms are optimized for all parameter values, they have the potential to adapt more gracefully to these changes. Also, the same code will adapt to varying input sizes forcing different memory levels to be in use. Finally, cache-oblivious algorithms automatically are optimizing the use of translation lookaside buffers (a cache holding recently accessed parts of the page table used for virtual memory) of the CPU, which may be seen as a second memory hierarchy parallel to the one mentioned in the introduction.

Possible weak points of the cache-oblivious model are the assumption of optimal off-line cache replacement, and the lack of modeling of the limited associativity of many of the levels of the hierarchy. The first point is mitigated by

the fact that normally, the provided analysis of a proposed cache-oblivious algorithm will work just as well assuming a Least-Recently-Used cache replacement policy, which is closer to actual replacement strategies of computers. The second point is also a weak point of most other memory models.

Key Results

This section surveys a number of the known results in the cache-oblivious model. Other surveys available include [5,14,20,24].

First of all, note that scanning an array of N elements takes $O(N/B)$ memory transfers for any values of B and M, and hence is an optimal cache-oblivious algorithm. Thus, standard RAM algorithms based on scanning may already possess good analysis in the cache-oblivious model – for instance, the classic deterministic selection algorithm has complexity $O(N/B)$ [20].

For sorting, a fundamental fact in the I/O-model is that comparison-based sorting of N elements takes $\Theta(\mathrm{Sort}(N))$ memory transfers [2], where $\mathrm{Sort}(N) = \frac{N}{B}\log_{M/B}\frac{N}{M}$. Also in the cache-oblivious model, sorting can be carried out in $\Theta(\mathrm{Sort}(N))$ memory transfer, if one makes the so-called *tall cache* assumption $M \geq B^{1+\varepsilon}$ [15,22]. Such an assumption has been shown to be necessary [16], which proves a separation in power between cache-oblivious algorithms and algorithms in the I/O-model (where this assumption is not needed for the sorting bound).

For searching, B-trees have cost $O(\log_B N)$, which is optimal in the I/O-model for comparison-based searching. This cost is also attainable in the cache-oblivious model, as shown for the static case in [25] and for the dynamic case in [13]. A number of later variants of cache-oblivious search trees have appeared. Also for searching, a separation between cache-oblivious algorithms and algorithms in the I/O-model has been shown [12] in the sense that the constants attainable in the $O(\log_B N)$ bound are provably different.

Permuting in the I/O-model has complexity $\Theta(\min\{\mathrm{Sort}(N), N\})$, assuming that elements are indivisible [2]. It has been proven [16] that this asymptotic complexity cannot be attained in the cache-oblivious model, hence also for this problem, a separation exists.

Cache-oblivious priority queues supporting operations in $O(1/B\log_{M/B} N/M)$ memory transfers amortized have been given.

Currently known cache-oblivious algorithms also include algorithms for problems in computational geometry [1,6,7,8,10,15], for graph problems [4,17,18,23], for scanning dynamic sets [9], for layout of static trees [11], for search problems on multi-sets [21], for dynamic programming [19], for partial persistence [10], for matrix operations [22], and for the Fast Fourier Transform [22].

Applications

The cache-oblivious model is a means for design and analysis of algorithms that use the memory hierarchy of computers efficiently.

Experimental Results

Cache-oblivious algorithms have been evaluated empirically in a number of areas, including sorting, searching, matrix algorithms [22], and dynamic programming [19]. The overall conclusion of these investigations is that cache-oblivious methods often outperform RAM algorithms, but not always exactly as much as do algorithms tuned to the specific memory hierarchy and problem size. On the other hand, cache-oblivious algorithms seem to perform well on all levels of the memory hierarchy, and to be more robust to changing problem sizes.

Cross References

- ► Cache-Oblivious B-Tree
- ► Cache-Oblivious Sorting
- ► I/O-model

Recommended Reading

1. Agarwal, P.K., Arge, L., Danner, A., Holland-Minkley, B.: Cache-oblivious data structures for orthogonal range searching. In: Proc. 19th ACM Symposium on Computational Geometry, pp. 237–245. ACM, New York (2003)
2. Aggarwal, A., Vitter, J.S.: The Input/Output complexity of sorting and related problems. Commun. ACM **31**(9), 1116–1127 (1988)
3. Arge, L.: External memory data structures. In: Abello, J., Pardalos, P.M., Resende, M.G.C. (eds.) Handbook of Massive Data Sets, pp. 313–358. Kluwer Academic Publishers, Boston (2002)
4. Arge, L., Bender, M.A., Demaine, E.D., Holland-Minkley, B., Munro, J.I.: Cache-oblivious priority queue and graph algorithm applications. In: Proc. 34th Annual ACM Symposium on Theory of Computing, pp. 268–276. ACM, New York (2002)
5. Arge, L., Brodal, G.S., Fagerberg, R.: Cache-oblivious data structures. In: Mehta, D., Sahni, S. (eds.) Handbook on Data Structures and Applications. CRC Press, Boca Raton (2005)
6. Arge, L., Brodal, G.S., Fagerberg, R., Laustsen, M.: Cache-oblivious planar orthogonal range searching and counting. In: Proc. 21st Annual ACM Symposium on Computational Geometry, pp. 160–169. ACM, New York (2005)
7. Arge, L., de Berg, M., Haverkort, H.J.: Cache-oblivious R-trees. In: Symposium on Computational Geometry, pp. 170–179. ACM, New York (2005)

8. Arge, L., Zeh, N.: Simple and semi-dynamic structures for cache-oblivious planar orthogonal range searching. In: Symposium on Computational Geometry, pp. 158–166. ACM, New York (2006)
9. Bender, M., Cole, R., Demaine, E., Farach-Colton, M.: Scanning and traversing: Maintaining data for traversals in a memory hierarchy. In: Proc. 10th Annual European Symposium on Algorithms. LNCS, vol. 2461, pp. 139–151. Springer, Berlin (2002)
10. Bender, M., Cole, R., Raman, R.: Exponential structures for cache-oblivious algorithms. In: Proc. 29th International Colloquium on Automata, Languages, and Programming. LNCS, vol. 2380, pp. 195–207. Springer, Berlin (2002)
11. Bender, M., Demaine, E., Farach-Colton, M.: Efficient tree layout in a multilevel memory hierarchy. In: Proc. 10th Annual European Symposium on Algorithms. LNCS, vol. 2461, pp. 165–173. Springer, Berlin (2002). Full version at http://arxiv.org/abs/cs/0211010
12. Bender, M.A., Brodal, G.S., Fagerberg, R., Ge, D., He, S., Hu, H., Iacono, J., López-Ortiz, A.: The cost of cache-oblivious searching. In: Proc. 44th Annual IEEE Symposium on Foundations of Computer Science, pp. 271–282. IEEE Computer Society Press, Los Alamitos (2003)
13. Bender, M.A., Demaine, E.D., Farach-Colton, M.: Cache-oblivious *B*-trees. In: 41st Annual Symposium on Foundations of Computer Science, pp. 399–409. IEEE Computer Society Press, Los Alamitos (2000)
14. Brodal, G.S.: Cache-oblivious algorithms and data structures. In: Proc. 9th Scandinavian Workshop on Algorithm Theory. LNCS, vol. 3111, pp. 3–13. Springer, Berlin (2004)
15. Brodal, G.S., Fagerberg, R.: Cache oblivious distribution sweeping. In: Proc. 29th International Colloquium on Automata, Languages, and Programming. LNCS, vol. 2380, pp. 426–438. Springer, Berlin (2002)
16. Brodal, G.S., Fagerberg, R.: On the limits of cache-obliviousness. In: Proc. 35th Annual ACM Symposium on Theory of Computing, pp. 307–315. ACM, New York (2003)
17. Brodal, G.S., Fagerberg, R., Meyer, U., Zeh, N.: Cache-oblivious data structures and algorithms for undirected breadth-first search and shortest paths. In: Proc. 9th Scandinavian Workshop on Algorithm Theory. LNCS, vol. 3111, pp. 480–492. Springer, Berlin (2004)
18. Chowdhury, R.A., Ramachandran, V.: Cache-oblivious shortest paths in graphs using buffer heap. In: Proc. 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures. ACM, New York (2004)
19. Chowdhury, R.A., Ramachandran, V.: Cache-oblivious dynamic programming. In: Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 591–600. ACM-SIAM, New York (2006)
20. Demaine, E.D.: Cache-oblivious algorithms and data structures. In: Proc. EFF summer school on massive data sets, LNCS. Springer, Berlin. To appear. Online version at http://theory.csail.mit.edu/edemaine/papers/BRICS2002/
21. Farzan, A., Ferragina, P., Franceschini, G., Munro, J.I.: Cache-oblivious comparison-based algorithms on multisets. In: Proc. 13th Annual European Symposium on Algorithms. LNCS, vol. 3669, pp. 305–316. Springer, Berlin (2005)
22. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache oblivious algorithms. In: 40th Annual IEEE Symposium on Foundations of Computer Science, pp. 285–298. IEEE Computer Society Press, Los Alamitos (1999)
23. Jampala, H., Zeh, N.: Cache-oblivious planar shortest paths. In: Proc. 32nd International Colloquium on Automata, Languages, and Programming. LNCS, vol. 3580, pp. 563–575. Springer, Berlin (2005)
24. Meyer, U., Sanders, P., Sibeyn, J.F. (eds.): Algorithms for Memory Hierarchies. LNCS, vol. 2625. Springer, Berlin (2003)
25. Prokop, H.: Cache-oblivious algorithms. Master's thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science (1999)
26. Vitter, J.S.: External memory algorithms and data structures: Dealing with MASSIVE data. ACM Comput. Surv. **33**(2), 209–271 (2001)
27. Vitter, J.S.: Geometric and spatial data structures in external memory. In: Mehta, D., Sahni, S. (eds.) Handbook on Data Structures and Applications. CRC Press, Boca Raton (2005)

Cache-Oblivious Sorting

1999; Frigo, Leiserson, Prokop, Ramachandran

Gerth Stølting Brodal
Department of Computer Science, University of Aarhus, Århus, Denmark

Keywords and Synonyms

Funnel sort

Problem Definition

Sorting a set of elements is one of the most well-studied computational problems. In the cache-oblivious setting the first study of sorting was presented in 1999 in the seminal paper by Frigo et al. [8] that introduced the cache-oblivious framework for developing algorithms aimed at machines with (unknown) hierarchical memory.

Model

In the cache-oblivious setting the computational model is a machine with two levels of memory: a cache of limited capacity and a secondary memory of infinite capacity. The capacity of the cache is assumed to be M elements and data is moved between the two levels of memory in blocks of B consecutive elements. Computations can only be performed on elements stored in cache, i. e. elements from secondary memory need to be moved to the cache before operations can access the elements. Programs are written as acting directly on one unbounded memory, i. e. programs are like standard RAM programs. The necessary block transfers between cache and secondary memory are handled automatically by the model, assuming an optimal offline cache replacement strategy. The core assumption of the cache-oblivious model is that M and B are *unknown to*

the algorithm whereas in the related I/O model introduced by Aggarwal and Vitter [1] the algorithms know M and B and the algorithms perform the block transfers explicitly. A thorough discussion of the cache-oblivious model and its relation to multi-level memory hierarchies is given in [8].

Sorting

For the sorting problem the input is an array of N elements residing in secondary memory, and the output is required to be an array in secondary memory storing the input elements in sorted order.

Key Results

In the I/O model tight upper and lower bounds were proved for the sorting problem and the problem of permuting an array [1]. In particular it was proved that sorting requires $\Theta(\frac{N}{B}\log_{M/B}\frac{N}{B})$ block transfers and permuting an array requires $\Theta(\min\{N, \frac{N}{B}\log_{M/B}\frac{N}{B}\})$ block transfers. Since lower bounds for the I/O model also hold for the cache-oblivious model, the lower bounds from [1] immediately give a lower bound of $\Omega(\frac{N}{B}\log_{M/B}\frac{N}{B})$ block transfers for cache-oblivious sorting and $\Omega(\min\{N, \frac{N}{B}\log_{M/B}\frac{N}{B}\})$ block transfers for cache-oblivious permuting. The upper bounds from [1] can not be applied to the cache-oblivious setting since these algorithms make explicit use of B and M.

Binary Mergesort performs $O(N\log_2 N)$ comparisons, but analyzed in the cache-oblivious model it performs $O(\frac{N}{B}\log_2\frac{N}{M})$ block transfers which is a factor $\Theta(\log\frac{M}{B})$ from the lower bound (assuming a recursive implementation of binary Mergesort, in order to get M in the denominator in the $\log N/M$ part of the bound on the block transfers). Another comparison-based sorting algorithm is the classical Quicksort sorting algorithm from 1962 by Hoare [9], that performs expected $O(N\log_2 N)$ comparisons and expected $O(\frac{N}{B}\log_2\frac{N}{M})$ block transfers. Both these algorithms achieve their relatively good performance for the number of block transfers from the fact that they are based on repeated scanning of arrays—a property not shared with e. g. Heapsort [10] that has a very poor performance of $\Theta(N\log_2\frac{N}{M})$ block transfers. In the I/O model the optimal performance of $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ is achieved by generalizing binary Mergesort to $\Theta(\frac{M}{B})$-way Mergesort [1].

Frigo et al. in [8] presented two cache-oblivious sorting algorithms (which can also be used to permute an array of elements). The first algorithm [8, Section 4] is denoted *Funnelsort* and is a reminiscent of classical binary Mergesort, whereas the second algorithm [8, Section 5] is a distribution-based sorting algorithm. Both algorithms perform *optimal* $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ block transfers – provided a *tall cache assumption* $M = \Omega(B^2)$ is satisfied.

Funnelsort

The basic idea of Funnelsort is to rearrange the sorting process performed by binary Mergesort, such that the processed data is stored "locally." This is achieved by two basic ideas: (1) A top-level recursion that partitions the input into $N^{1/3}$ sequences of size $N^{2/3}$, Funnelsort these sequences recursively, and merge the resulting sorted subsequences using an $N^{1/3}$-*merger*. (2) A k-merger is recursively defined to perform binary merging of k input sequences in a clever schedule with an appropriate recursive layout of data in memory using buffers to hold suspended merging processes (see Fig. 1). Subsequently two simplifications were made, without sacrificing the asymptotic number of block transfers performed. In [3] it was proved that the binary merging could be performed lazily, simplifying the scheduling of merging. In [5] it was further observed that the recursive layout of k-mergers is not necessary. It is sufficient that a k-merger is stored in a consecutive array, i. e. the buffers can be laid out in arbitrary order which simplifies the construction algorithm for the k-merger.

Implicit Cache-Oblivious Sorting

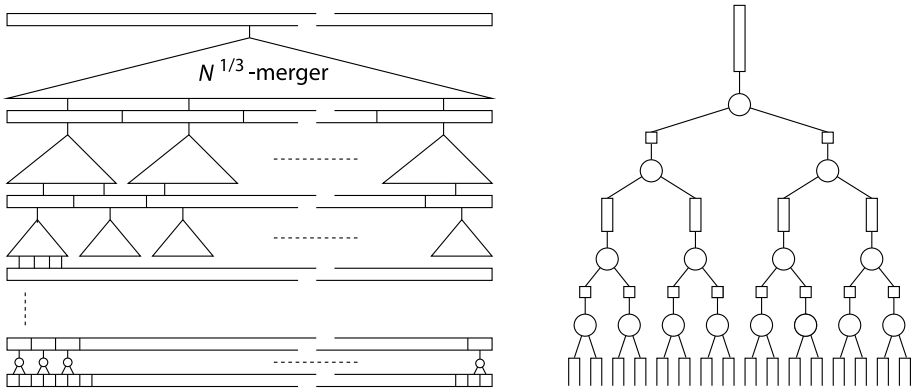
Franceschini in [7] showed how to perform optimal cache-oblivious sorting implicitly using only $O(1)$ space, i. e. all data is stored in the input array except for $O(1)$ additional words of information. In particular the output array is just a permutation of the input array.

The Role of the Tall Cache Assumption

The role of the tall cache assumption on cache-oblivious sorting was studied by Brodal and Fagerberg in [4]. If no tall cache assumption is made, they proved the following theorem:

Theorem 1 ([4], Corollary 3) *Let $B_1 = 1$ and $B_2 = M/2$. For any cache-oblivious comparison-based sorting algorithm, let t_1 and t_2 be upper bounds on the number of I/Os performed for block sizes B_1 and B_2. If for a real number $d \geq 0$ it is satisfied that $t_2 = d \cdot \frac{N}{B_2}\log_{M/B_2}\frac{N}{B_2}$ then $t_1 > 1/8 \cdot N\log_2 N/M$.*

The theorem shows cache-oblivious comparison-based sorting without a tall cache assumption cannot match the performance of algorithms in the I/O model where M and



Cache-Oblivious Sorting, Figure 1
The overall recursion of Funnelsort (*left*) and a 16-merger (*right*)

B are known to the algorithm. It also has the natural interpretation that if a cache-oblivious algorithm is required to be I/O-optimal for the case $B = M/2$, then binary Mergesort is best possible –any other algorithm will be the same factor of $\Theta(\log M)$ worse than the optimal block transfer bound for the case $M \gg B$.

For the related problem of permuting an array the following theorem states that for all possible tall cache assumptions $B \leq M^{\delta}$, no cache-oblivious permuting algorithm exists with a block transfer bound (even only in the average case sense) matching the worst case bound in the I/O model.

Theorem 2 ([4], Theorem 2) *For all $\delta > 0$, there exists no cache-oblivious algorithm for permuting that for all $M \geq 2B$ and $1 \leq B \leq M^{\delta}$ achieves $O(\min\{N, \frac{N}{B} \log_{M/B} \frac{N}{B}\})$ I/Os averaged over all possible permutations of size N.*

Applications

Many problems can be reduced to cache-oblivious sorting. In particular Arge et al. [2] developed a cache-oblivious priority queue based on a reduction to sorting. They furthermore showed how a cache-oblivious priority queue can be applied to solve a sequence of graph problems, including list ranking, BFS, DFS, and minimum spanning trees.

Brodal and Fagerberg in [3] showed how to modify the cache-oblivious lazy Funnelsort algorithm to solve several problems within computational geometry, including orthogonal line segment intersection reporting, all nearest neighbors, 3D maxima problem, and batched orthogonal range queries. All these problems can be solved by a computation process very similarly to binary Mergesort with an additional problem dependent twist. This general framework to solve computational geometry problems is denoted *distribution sweeping*.

Open Problems

Since the seminal paper by Frigo et al. [8] introducing the cache-oblivious framework there has been a lot of work on developing algorithms with a good theoretical performance, but only a limited amount of work has been done on implementing these algorithms. An important issue for future work is to get further experimental results consolidating the cache-oblivious model as a relevant model for dealing efficiently with hierarchical memory.

Experimental Results

A detailed experimental study of the cache-oblivious sorting algorithm Funnelsort was performed in [5]. The main result of [5] is that a carefully implemented cache-oblivious sorting algorithm can be faster than a tuned implementation of Quicksort already for input sizes well within the limits of RAM. The implementation is also at least as fast as the recent cache-aware implementations included in the test. On disk the difference is even more pronounced regarding Quicksort and the cache-aware algorithms, whereas the algorithm is slower than a careful implementation of multiway Mergesort optimized for external memory such as in TPIE [6].

URL to Code

http://kristoffer.vinther.name/projects/funnelsort/

Cross References

- ► Cache-Oblivious Model
- ► External Sorting and Permuting
- ► I/O-model

Recommended Reading

1. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Commun. ACM **31**(9), 1116–1127 (1988)
2. Arge, L., Bender, M.A., Demaine, E.D., Holland-Minkley, B., Munro, J.I.: Cache-oblivious priority queue and graph algorithm applications. In: Proc. 34th Annual ACM Symposium on Theory of Computing, pp. 268–276. ACM Press, New York (2002)
3. Brodal, G.S., Fagerberg, R.: Cache oblivious distribution sweeping. In: Proc. 29th International Colloquium on Automata, Languages, and Programming. Lecture Notes in Computer Science, vol. 2380, pp. 426–438. Springer, Berlin (2002)
4. Brodal, G.S., Fagerberg, R.: On the limits of cache-obliviousness. In: Proc. 35th Annual ACM Symposium on Theory of Computing, pp. 307–315. ACM Press, New York (2003)
5. Brodal, G.S., Fagerberg, R., Vinther, K.: Engineering a cache-oblivious sorting algorithm. ACM J. Exp. Algoritmics (Special Issue of ALENEX 2004) **12**(2.2), 23 (2007)
6. Department of Computer Science, Duke University. TPIE: a transparent parallel I/O environment. http://www.cs.duke.edu/TPIE/. Accessed 2002
7. Franceschini, G.: Proximity mergesort: Optimal in-place sorting in the cache-oblivious model. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, p. 291. Philadelphia, 2004
8. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: Proc. 40th Annual Symposium on Foundations of Computer Science, pp. 285–297. IEEE Computer Society Press, Los Alamitos (1999)
9. Hoare, C.A.R.: Quicksort. Comput. J. **5**(1), 10–15 (1962)
10. Williams, J.W.J.: Algorithm 232: Heapsort. Commun. ACM **7**(6), 347–348 (1964)

Caching

- ► Online Paging and Caching
- ► Paging

Causal Order, Logical Clocks, State Machine Replication

1978; Lamport

Xavier Défago
School of Information Science, Japan Advanced Institute of Science and Technology (JAIST),
Ishikawa, Japan

Keywords and Synonyms

State-machine replication: active replication

Problem Definition

This entry covers several problems, related with each other. The first problem is concerned with maintaining the causal relationship between events in a distributed system. The motivation is to allow distributed systems to reason about time with no explicit access to a physical clock. Lamport [5] defines a notion of logical clocks that can be used to generate timestamps that are consistent with causal relationships (in a conservative sense). He illustrates logical clocks (also called Lamport clocks) with a distributed mutual exclusion algorithm. The algorithm turns out to be an illustration of state-machine replication. Basically, the algorithm generates a total ordering of the events that is consistent across processes. With all processes starting in the same state, they evolve consistently with no need for further synchronization.

System Model

The system consists of a collection of processes. Each process consists of a sequence of events. Processes have no shared memory and communicate only by exchanging messages. The exact definition of an event depends on the system actually considered and the abstraction level at which it is considered. One distinguishes between three kinds of events: internal (affects only the process executing it), send, and receive events.

Causal Order

Causal order is concerned with the problem that the occurrence of some events may affect other events in the future, while other events may not influence each other. With processes that do not measure time, the notion of simultaneity must be redefined in such a way that simultaneous events are those that cannot possibly affect each other. For this reason, it is necessary to define what it means for an event to happen before another event.

The following "happened before" relation is defined as an irreflexive partial ordering on the set of all events in the system [5].

Definition 1 The relation "$\rightarrow$" on the set of events of a system is the smallest relation satisfying the following three conditions:

1. If a and b are events in the same process, and a comes before b, then $a \rightarrow b$.

2. If a is the sending of a message by one process and b is the receipt of the same message by another process, then $a \rightarrow b$.
3. If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$.

Definition 2 Two distinct events a and b are said to be *concurrent* if $a \not\rightarrow b$ and $b \not\rightarrow a$.

Logical Clocks

Lamport also defines clocks in a generic way, as follows.

Definition 3 A clock C_i for a process p_i is a function which assigns a number $C_i\langle a\rangle$ to any event a on that process. The entire system of clocks is represented by the function C which assigns to any event b the number $C\langle b\rangle$, where $C\langle b\rangle = C_j\langle b\rangle$ if b is an event in process p_j. The system of clocks must meet the following *clock condition*.

- For any events a and b, if $a \rightarrow b$ then $C\langle a\rangle < C\langle b\rangle$.

Assuming that there is some arbitrary total ordering $\prec$ of the processes (e. g., unique names ordered lexicographically), Lamport extends the "happened before" relation and defines a relation "$\Rightarrow$" as a total ordering on the set of all events in the system.

Definition 4 The total order relation $\Rightarrow$ is defined as follows. If a is an event in process p_i and b is an event in process p_j, then $a \Rightarrow b$ if and only if either one of the following conditions is satisfied.

1. $C_i\langle a\rangle < C_j\langle b\rangle$
2. $C_i\langle a\rangle = C_j\langle b\rangle$ and $p_i \prec p_j$.

In fact, Lamport [5] also discusses an adaptation of these conditions to physical clocks, and provides a simple clock synchronization algorithm. This is however not discussed further here.

State Machine Replication

The problem of state-machine replication was originally presented by Lamport [4,5]. In a later review of the problem, Schneider [8] defines the problem as follows (formulation adapted to the context of the entry).

Problem 1 (State-machine replication)
INPUT: *A set of concurrent requests.*
OUTPUT: *A sequence of the requests processed at each process, such that:*

1. *Replica coordination: all replicas receive and process the same sequence of requests.*
2. *Agreement: every non-faulty state-machine replica receives every request.*
3. *Order: every non-faulty state-machine replica processes the requests it receives in the same relative order.*

In his paper on logical time [5] and discussed in this entry, Lamport does not consider failures. He does however consider them in another paper on state-machine replication for fault-tolerance [4], which he published the same year.

Key Results

Lamport [5] proposed many key results related to the problems described above.

Logical Clocks

Lamport [5] defines an elegant system of logical clocks that meets the clock condition presented in Definition 3. The clock of a process p_i is represented by a register C_i, such that $C_i\langle a\rangle$ is the value held by C_i when a occurs. Each message m carries a timestamp T_m, which equals the time at which m was sent. The clock system can be described in terms of the following rules.

1. Each process p_i increments C_i between any two successive events.
2. If event a is the sending of a message m by process p_i, then the message m contains a timestamp $T_m = C_i\langle a\rangle$.
3. Upon receiving a message m, process p_j sets C_j to $\max(C_j, T_m + 1)$ (before actually executing the receive event).

State Machine Replication

As an illustration for the use of logical clocks, Lamport [5] describes a mutual exclusion algorithm. He also mentions that the approach is more general and discusses the concept of state-machine replication that he refines in a different paper [4].

The mutual exclusion algorithm is based on the idea that every process maintains a copy of a request queue, and the algorithm ensures that the copies remain consistent across the processes. This is done by generating a total ordering of the request messages, according to timestamps obtained from the logical clocks of the sending processes.

The algorithm described works under the following simplifying assumptions:

- Every message that is sent is eventually received.
- For any processes p_i and p_j, messages from p_i to p_j are received in the same order as they are sent.
- A process can send messages directly to every other processes.

The algorithm requires that each process maintains its own request queue, and ensures that the request queues of different processes always remain consistent. Initially, request queues contain a single message $(T_0, p_0, \mathit{request})$, where p_0 is the process that holds the resource and the

timestamp T_0 is smaller than the initial value of every clock. Then, the algorithm works as follows.

1. When a process p_i requests the resource, it sends a request message $(T_m, p_i, request)$ to all other processes and puts the message in its request queue.
2. When a process p_j receives a message $(T_m, p_i, request)$, it puts that message in its request queue and sends an acknowledgment $(T_{m'}, p_j, ack)$ to p_i.
3. When a process p_i releases the resource, it removes all instances of messages $(-, p_i, request)$ from its queue, and sends a message $(T_{m'}, p_i, release)$ to all other processes.
4. When a process p_j receives a release message from process p_i, it removes all instances of messages $(-, p_i, request)$ from its queue, and sends a timestamped acknowledgment to p_i.
5. Messages in the queue are sorted according to the total order relation $\Rightarrow$ of Definition 4. A process p_i can use the resource when (a) a message $(T_m, p_i, request)$ appears first in the queue, and (b) p_i has received from all other processes a message with a timestamp greater than T_m (or equal from any process p_j where $p_i \prec p_j$).

Applications

A brief overview of some applications of the concepts presented in this entry has been provided.

First of all, the notion of causality in distributed systems (or lack thereof) leads to a famous problem in which a user may potentially see an answer before she can see the relevant question. The time-independent characterization of causality of Lamport lead to the development of efficient solutions to enforce causal order in communication. In his later work, Lamport [3] gives a more general definition to the "happened before" relation, so that a system can be characterized at various abstraction levels.

About a decade after Lamport's work on logical clock, Fidge [2] and Mattern [6] have developed the notion of vector clocks, with the advantage of a complete characterization of causal order. Indeed, the clock condition enforced by Lamport's logical clocks is only a one-way implication (see Definition 3). In contrast, vector clocks extend Lamport clocks by ensuring that, for any events a and b, if $C\langle a\rangle < C\langle b\rangle$, then $a \rightarrow b$. This is for instance useful for choosing a set of checkpoints after recovery of a distributed system, for distributed debugging, or for deadlock detection. Other extensions of logical time have been proposed, that have been surveyed by Raynal and Singhal [7].

The state-machine replication also has many applications. In particular, it is often used for replicating a distributed service over several processors, so that the service can continue to operate even in spite of the failure of some of the processors. State-machine replication ensures that the different replicas remain consistent.

The mutual exclusion algorithm proposed by Lamport [5] and described in this entry is actually one of the first known solution to the *atomic broadcast* problem (see relevant entry). Briefly, in a system with several processes that broadcast messages concurrently, the problem requires that all processes deliver (and process) all message in the same order. Nowadays, there exist several approaches to solving the problem. Surveying many algorithms, Défago et al. [1] have classified Lamport's algorithm as *communication history* algorithms, because of the way the ordering is generated.

Cross References

- ▶ Atomic Broadcast
- ▶ Clock Synchronization
- ▶ Concurrent Programming, Mutual Exclusion
- ▶ Linearizability
- ▶ Quorums

Recommended Reading

1. Défago, X., Schiper, A., Urbán, P.: Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Comput. Surv. **36**, 372–421 (2004)
2. Fidge, C. J.: Logical time in distributed computing systems. IEEE Comput. **24**, 28–33 (1991)
3. Lamport, L.: On interprocess communication. Part I: Basic formalism. Distrib. Comput. **1**, 77–85 (1986)
4. Lamport, L.: The implementation of reliable distributed multiprocess systems. Comput. Netw. **2**, 95–114 (1978)
5. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**, 558–565 (1978)
6. Mattern, F.: Virtual time and global states of distributed systems. In: Cosnard, M., Quinton, P. (eds.) Parallel and Distributed Algorithms, pp.215–226. North-Holland, Amsterdam (1989)
7. Raynal, M., Singhal, M.: Capturing causality in distributed systems. IEEE Comput. **29**, 49–56 (1996)
8. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surv. **22**, 299–319 (1990)

Certificate Complexity and Exact Learning

1995; Hellerstein, Pilliapakkamnatt, Raghavan, Wilkins

Lisa Hellerstein
Department of Computer and Information Science, Polytechnic University, Brooklyn, NY, USA

Problem Definition

This problem concerns the query complexity of proper learning in a widely studied learning model: exact learning with membership and equivalence queries. Hellerstein et al. [8] showed that the number of (polynomially sized) queries required to learn a concept class in this model is closely related to the size of certain certificates associated with that class. This relationship gives a combinatorial characterization of the concept classes that can be learned with polynomial query complexity. (Similar results were shown by Hegedüs[7], based on the work of Moshkov [11].)

The Exact Learning Model

Concepts are functions $f : X \to Y$ where X is an arbitrary domain, and $Y = \{0, 1\}$. In exact learning, there is a hidden concept f from a known class of concepts C, and the problem is to exactly identify the function f.

Algorithms in the exact learning model obtain information about f, the target concept, by querying two oracles, a membership oracle and an equivalence oracle. A membership oracle for f answers membership queries, which are of the form "What is $f(x)$?" where $x \in X$ (point evaluation queries). The membership oracle responds with the value $f(x)$. An equivalence oracle for f answers equivalence queries, which are of the form "Is $h \equiv f$?" where h is a representation of a concept defined on the domain X. Representation h is called a hypothesis. The equivalence oracle responds "yes" if $h(x) = f(x)$ for all $x \in X$. Otherwise, it returns a counterexample, a value $x \in X$ such that $f(x) \neq h(x)$.

The exact learning model is due to Angluin [2]. Angluin viewed the combination of membership and equivalence oracles as constituting a "minimally adequate teacher." Equivalence queries can be simulated both in Valiant's well-known PAC model, and in the on-line mistake-bound learning model.

Let R be a set of representations of concepts, and let C_R be the associated set of concepts. For example, if R were a set of DNF formulas, then C_R would be the set of Boolean functions (concepts) represented by those formulas. An exact learning algorithm is said to learn R if, given access to membership and equivalence oracles for any f in C_R, it ends by outputing a hypothesis h that is a representation of f.

Query Complexity of Exact Learning

There are two aspects to the complexity of exact learning, query complexity and computational complexity. The results of Hellerstein et al. concern query complexity.

The query complexity of an exact learning algorithm measures the number of queries it asks and the size of the hypotheses it uses in those queries (and as the final output). We assume that each representation class R has an associated size function that assigns a non-negative number to each $r \in R$. The size of a concept c with respect to R, denoted by $|c|_R$, is the size of the smallest representation of c in R; if $c \notin c_R$, $|c|_R = \infty$. Ideally, the query complexity of an exact learning algorithm will be polynomial in the size of the target and other relevant parameters of the problem.

Many exact learning results concern learning representations of Boolean functions. Algorithms for learning such classes R are said to have polynomial query complexity if the number of hypotheses used, and the size of those hypotheses, is bounded by some polynomial $p(m, n)$, where n is the number of variables on which the target f is defined, and $m = |f|_R$. We assume that algorithms for learning Boolean representation classes are given the value of n as input.

Since the number and size of queries used by an algorithm is a lower bound on the time taken by that algorithm, query complexity lower bounds imply computational complexity lower bounds.

Improper Learning and the Halving Algorithm

An algorithm for learning a representation class R is said to be proper if all hypotheses used in its equivalence queries are from R_C, and it outputs a representation from R_C. Otherwise, the algorithm is said to be improper.

When R_C is a finite concept class, defined on a finite domain X, a simple, generic algorithm called the halving algorithm can be used to exactly learn R using $\log |R_C|$ equivalence queries and no membership queries. The halving algorithm is based on the following idea. For any $V \subseteq R_C$, define the majority hypothesis MAJ_V to be the concept defined on X such that for all $x \in X$, $MAJ_V(x) = 1$ if $g(x) = 1$ for more than half the concepts g in V, and $MAJ_V(x) = 0$ otherwise. The halving algorithm begins by setting $V = R_C$. It then repeats the following:

1. Ask an equivalence query with the hypothesis MAJ_V.
2. If the answer is yes, then output MAJ_V.
3. Otherwise, the answer is a counterexample x. Remove from V all g such that $g(x) = MAJ_V(x)$.

Each counterexample eliminates the majority of the elements currently in V, so the size of V is reduced by a factor of at least 2 with each equivalence query. It follows that the algorithm cannot ask more than $\log_2 |R_C|$ queries.

The halving algorithm cannot necessarily be implemented as a proper algorithm, since the majority hypotheses may not be representable in R_C. Even when they are

representable in R_C, the representations may be exponentially larger than the target concept.

Proper Learning and Certificates

In the exact model, the query complexity of proper learning is closely related to the size of certain certificates.

For any concept f defined on a domain X, a certificate that f has property P is a subset $S \subseteq X$ such that for all concepts g defined on X, if $g(x) = f(x)$ for all $x \in X$, then g has property P. The size of the certificate S is $|S|$, the number of elements in it.

We are interested in properties of the form "g is not a member of the concept class C". To take a simple example, let D be the class of constant-valued n-variable Boolean functions, i. e. D consists of the two functions $f_1(x_1, \ldots, x_n) = 1$ and $f_2(x_1, \ldots, x_n) = 0$. Then if g is an n-variable Boolean function that is not a member of D, a certificate that g is not in C could be just a pair $a \in \{0,1\}^n$ and $b \in \{0,1\}^n$ such that $g(a) = 1$ and $g(b) = 0$.

For C a class of concepts defined on X, define the exclusion dimension of C to be the maximum, over all concepts g not in C, of the size of the smallest certificate that g is not in C. Let $XD(C)$ denote the exclusion dimension of C. In the above example, $XD(C) = 2$.

Key Results

Theorem 1 *Let R be a finite class of representations. Then there exists a proper learning algorithm in the exact model that learns C using at most $XD(C) \log |C|$ queries. Further, any such algorithm for C must make at least $XD(C)$ queries.*

Independently, Hegedüs proved a theorem that is essentially identical to the above theorem. The algorithm in the theorem is a variant of the ordinary halving algorithm. As noted by Hegedüs, a similar result to Theorem 1 was proved earlier by Moshkov, and Moshkov's techniques can be used to improve the upper bound by a factor of $1/XD(C)$.

An extension of the above result characterizes the representation classes that have polynomial query complexity. The following theorem presents the extended result as it applies to representation classes of Boolean functions.

Theorem 2 *Let R be a class of representations of Boolean functions. Then there exists a proper exact learning algorithm for R with polynomial query complexity iff there exists a polynomial $p(m, n)$ such that for all $m, n > 0$, and all n-variable Boolean functions g, if the size of g is greater than m, then there exists a certificate of size at most $p(m, n)$ proving that $|g|_R > m$.*

A concept class having certificates of the type specified in this theorem is said to have polynomial certificates.

The algorithm in the above theorem does not run in polynomial time. Hellerstein et al. give a more complex algorithm that runs in polynomial time using a Σ_4^P oracle, provided R satisfies certain technical conditions. Köbler and Lindner subsequently gave an algorithm using a Σ_2^P oracle [10].

Theorem 2 and its generalization give a technique for proving bounds on proper learning in the exact model. Proving upper bounds on the size of the appropriate certificates yields upper bounds on query complexity. Proving lower bounds on the size of appropriate certificates yields lower bounds on query complexity and hence also on time complexity. Moreover, unlike many computational hardness results in learning, computational hardness results achieved in this way do not rely on any unproven complexity theoretic or cryptographic hardness assumptions.

One of the most widely studied problems in computational learning theory has been the question of whether DNF formulas can be learned in polynomial time in common learning models. The following result on learning DNF formulas was proved using Theorem 2, by bounding the size of the relevant certificates.

Theorem 3 *There is a proper algorithm that learns DNF formulas in the exact model with query complexity bounded above by a polynomial $p(m, r, n)$, where m is the size of the smallest DNF representing the target function f, n is the number of variables on which f is defined, and r is the size of the smallest CNF representing f.*

The size of a DNF is the number of its terms; the size of a CNF is the number of its clauses. The above theorem does not imply polynomial-time learnability of arbitrary DNF formulas, since the running time of the algorithm depends not just on the size of the smallest DNF representing the target, but also on the size of the smallest CNF.

Building on results of Alekhnovich et al., Feldman showed that if NP $\neq$ RP, DNF formulas cannot be learned in polynomial time in the PAC model augmented with membership queries. The same negative result then follows immediately for the exact model [1,6]. Hellerstein and Raghavan used certificate size lower bounds and Theorem 1 to prove that DNF formulas cannot be learned by a proper exact algorithm with polynomial query complex-

ity, if the algorithm is restricted to using DNF hypotheses that are only slightly larger than the target [9].

The main results of Hellerstein et al. apply to learning with membership and equivalence queries. Hellerstein et al. also considered the model of exact learning with membership queries alone, and showed that in this model, a projection-closed Boolean function class is polynomial query learnable iff it has polynomial teaching dimension. Teaching dimension was previously defined by Goldman and Kearns. Hegedüs defined the extended teaching dimension, and showed that all classes are polynomially query learnable with membership queries alone iff they have polynomial extended teaching dimension.

Balcázar et al. introduced the strong consistency dimension to characterize polynomial query learnability with equivalence queries alone [5]. The abstract identification dimension of Balcázar, Castro, and Guijarro is a very general measure that can be used to characterize polynomial query learnability for any set of example-based queries [4].

Applications

None

Open Problems

It remains open whether DNF formulas can be learned in polynomial time in the exact model, using hypotheses that are not DNF formulas.

Feldman's results show the computational hardness of proper learning of DNF in the exact learning model based on complexity theoretic assumptions. However, it is unclear whether the hardness of proper learning of DNF is a result of computational barriers, or whether query complexity is also a barrier to efficient learning. It is still open whether the class of DNF formulas has polynomial certificates; showing they do not have polynomial certificates would give a hardness result for proper learning of DNF based only on query complexity, with no complexity theoretic assumptions (and without the hypothesis-size restrictions used by Hellerstein and Raghavan). It is also open whether the class of Boolean decision trees has polynomial certificates.

Certificate techniques are used to prove lower bounds on learning when we restrict the type of hypotheses used by the learning algorithm. These types of results are called representation dependent, since they depend on the restriction of the representations used as hypotheses. Although there are some techniques for proving representation-independent hardness results, there is a need for more powerful techniques.

Recommended Reading

1. Alekhnovich, M., Braverman, M., Feldman, V., Klivans, A.R., Pitassi, T.: Learnability and automatizability. In: FOCS '04 Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), pp. 621–630. IEEE Computer Society, Washington (2004)
2. Angluin, D.: Queries and concept learning. Mach. Learn. **2**(4), 319–342 (1987)
3. Angluin, D.: Queries Revisited. Theor. Comput. Sci. **313**(2), 175–194 (2004)
4. Balcázar, J.L., Castro, J., Guijarro, D.: A new abstract combinatorial dimension for exact learning via queries. J. Comput. Syst. Sci. **64**(1), 2–21 (2002)
5. Balcázar, J.L., Castro, J., Guijarro, D., Simon, H.-U.: The consistency dimension and distribution-dependent learning from queries. Theor. Comput. Sci. **288**(2), 197–215 (2002)
6. Feldman, V.: Hardness of approximate two-level logic minimization and pac learning with membership queries. In: STOC '06 Proceedings of the 38th Annual ACM Symposium on Theory of Computing, pp. 363–372. ACM Press, New York (2006)
7. Hegedüs, T.: Generalized teaching dimensions and the query complexity of learning. In: COLT '95 Proceedings of the 8th Annual Conference on Computational Learning Theory, pp. 108–117 (1995)
8. Hellerstein, L., Pillaipakkamnatt, K., Raghavan, V., Wilkins, D.: How many queries are needed to learn? J. ACM. **43**(5), 840–862 (1996)
9. Hellerstein, L., Raghavan, V.: Exact learning of dnf formulas using dnf hypotheses. J Comput. Syst. Sci. **70**(4), 435–470 (2005)
10. Köbler, J., Lindner, W.: Oracles in s^P_2 are sufficient for exact learning. Int. J. Found. Comput. Sci. **11**(4), 615–632 (2000)
11. Moshkov, M.Y.: Conditional tests. Probl. Kibern. (in Russian) **40**, 131–170 (1983)

Channel Assignment and Routing in Multi-Radio Wireless Mesh Networks

2005; Alicherry, Bhatia, Li

Mansoor Alicherry, Randeep Bhatia, Li (Erran) Li
Bell Labs, Alcatel-Lucent, Murray Hill, NJ, USA

Keywords and Synonyms

Graph coloring, Ad-hoc networks

Problem Definition

One of the major problems facing wireless networks is the capacity reduction due to interference among multiple simultaneous transmissions. In wireless mesh networks providing mesh routers with multiple-radios can greatly alleviate this problem. With multiple-radios, nodes can transmit and receive simultaneously or can transmit on

multiple channels simultaneously. However, due to the limited number of channels available the interference cannot be completely eliminated and in addition careful channel assignment must be carried out to mitigate the effects of interference. Channel assignment and routing are inter-dependent. This is because channel assignments have an impact on link bandwidths and the extent to which link transmissions interfere. This clearly impacts the routing used to satisfy traffic demands. In the same way traffic routing determines the traffic flows for each link which certainly affects channel assignments. Channel assignments need to be done in a way such that the communication requirements for the links can be met. Thus, the problem of throughput maximization of wireless mesh networks must be solved through channel assignment, routing, and scheduling.

Formally, given a wireless mesh backbone network modeled as a graph (V, E): The node $t \in V$ represents the wired network. An edge $e = (u, v)$ exists in E iff u and v are within *communication range* R_T. The set $V_G \subseteq V$ represents the set of gateway nodes. The system has a total of K channels. Each node $u \in V$ has $I(u)$ network interface cards, and has an aggregated demand $l(u)$ from its associated users. For each edge e the set $I(e) \subset E$ denotes the set of edges that it interferes with. A pair of nodes that use the same channel and are within *interference range* $R_I x$ may interfere with each other's communication, even if they cannot directly communicate. Node pairs using different channels can transmit packets simultaneously without interference. The problem is to maximize λ where at least $\lambda l(u)$ amount of throughput can be routed from each node u to the Internet (represented by a node t). The $\lambda l(u)$ throughput for each node u is achieved by computing $g(1)$ a network flow that associates with each edge $e = (u, v)$ values $f(e(i)), 1 \leq i \leq K$ where $f(e(i))$ is the rate at which traffic is transmitted by node u for node v on channel i; (2) a feasible channel assignment $F(u)$ ($F(u)$ is an ordered set where the ith interface of u operates on the ith channel in $F(u)$) such that, whenever $f(e(i)) > 0$, $i \in F(u) \cap F(v)$; (3) a *feasible* schedule S that decides the set of edge channel pair (e, i) (edge e using channel, i. e. $f(e(i)) > 0$ scheduled at time slot τ, for $\tau = 1, 2, \ldots, T$ where T is the period of the schedule. A schedule is feasible if the edges of no two edge pairs $(e_1, i), (e_2, i)$ scheduled in the same time slot for a common channel i interfere with each other ($e_1 \notin I(e_2)$ and $e_2 \notin I(e_1)$). Thus, a feasible schedule is also referred to as an interference free edge schedule. An indicator variable $X_{e,i,\tau}$, $e \in E$, $i \in F(e)$, $\tau \geq 1$ is used. It is assigned 1 if and only if link e is active in slot τ on channel i. Note that $1/T \sum_{1 \leq \tau \leq T} X_{e,i,\tau} c(e) = f(e(i))$. This is because communication at rate $c(e)$ happens in every slot that link e is active on channel i and since $f(e(i))$ is the average rate attained on link e for channel i. This implies $1/T \sum_{1 \leq \tau \leq T} X_{e,i,\tau} = \frac{f(e(i))}{c(e)}$.

Joint Routing, Channel Assignment, and Link Scheduling Algorithm

Even the interference free edge scheduling sub-problem given the edge flows is NP-hard [5]. An approximation algorithm called RCL for the joint routing, channel assignment, and link scheduling problem has been developed. The algorithm performs the following steps in the given order:

1. **Solve LP:** First optimally solve a LP relaxation of the problem. This results in a flow on the flow graph along with a not necessarily feasible channel assignment for the node radios. Specifically, a node may be assigned more channels than the number of its radios. However, this channel assignment is "optimal" in terms of ensuring that the interference for each channel is minimum. This step also yields a lower bound on the λ value which is used in establishing the worst case performance guarantee of the overall algorithm.
2. **Channel Assignment:** This step presents a channel assignment algorithm which is used to adjust the flow on the flow graph (routing changes) to ensure a feasible channel assignment. This flow adjustment also strives to keep the increase in interference for each channel to a minimum.
3. **Interference Free Link Scheduling:** This step obtains an interference free link schedule for the edge flows corresponding to the flow on the flow graph.

Each of these steps is described in the following subsections.

A Linear Programming-Based Routing Algorithm

A linear program LP (1) to find a flow that maximizes λ is given below:

$$\max \lambda \tag{1}$$

Subject to

$$\lambda l(v) + \sum_{e=(u,v) \in E} \sum_{i=1}^{K} f(e(i)) = \sum_{e=(v,u) \in E} \sum_{i=1}^{K} f(e(i)), \quad \forall v \in V - V_G \tag{2}$$

$$f(e(i)) \leq c(e), \quad \forall e \in E \tag{3}$$

$$\sum_{1\le i\le K}\left(\sum_{e=(u,v)\in E}\frac{f(e(i))}{c(e)}+\sum_{e=(v,u)\in E}\frac{f(e(i))}{c(e)}\right)\le I(v)\,,\quad v\in V \qquad (4)$$

$$\frac{f(e(i))}{c(e)}+\sum_{e'\in I(e)}\frac{f(e'(i))}{c(e')}\le c(q)\,,\quad \forall e\in E\,,\ 1\le i\le K\,. \qquad (5)$$

The first two constraints are *flow constraints*. The first one is the flow conservation constraint; the second one ensures no link capacity is violated. The third constraint is the *node radio constraints*. Recall that a IWMN node $v \in V$ has $I(v)$ radios and hence can be assigned at most $I(v)$ channels from $1 \le i \le K$. One way to model this constraint is to observe that due to interference constraints v can be involved in at most $I(v)$ simultaneous communications (with different one hop neighbors). In other words this constraint follows from $\sum_{1\le i\le K}\sum_{e=(u,v)\in E} X_{e,i,\tau} + \sum_{1\le i\le K}\sum_{e=(v,u)\in E} X_{e,i,\tau} \le I(v)$. The fourth constraint is the *link congestion constraints* which are discussed in detail in Sect. "Link Flow Scheduling". Note that all the constraints listed above are necessary conditions for any feasible solution. However, these constraints are not necessarily sufficient. Hence if a solution is found that satisfies these constraints it may not be a feasible solution. The approach is to start with a "good" but not necessarily feasible solution that satisfies all of these constraints and use it to construct a feasible solution without impacting the quality of the solution.

A solution to this LP can be viewed as a flow on a *flow graph* $H = (V, E^H)$ where $E^H = \{e(i)|\forall e \in E, 1 \le i \le K\}$. Although the optimal solution to this LP yields the best possible λ (say λ^*) from a practical point of view more improvements may be possible:

- The flow may have directed cycles. This may be the case since the LP does not try to minimize the amount of interference directly. By removing the flow on the directed cycle (equal amount off each edge) flow conservation is maintained and in addition since there are fewer transmissions the amount of interference is reduced.
- The flow may be using a long path when shorter paths are available. Note that longer paths imply more link transmissions. In this case it is often the case that by moving the flow to shorter paths, system interference may be reduced.

The above arguments suggest that it would be practical to find among all solutions that attain the optimal λ value of λ^* the one for which the total value of the following quantity is minimized:

$$\sum_{1\le i\le K}\sum_{e=(v,u)\in E}\frac{f(e(i))}{c(e)}.$$

The LP is then re-solved with this objective function and with λ fixed at λ^*.

Channel Assignment

The solution to the LP (1) is a set of flow values $f(e(i))$ for edge e and channel i that maximize the value λ. Let λ^* denote the optimal value of λ. The flow $f(e(i))$ implies a channel assignment where the two end nodes of edge e are both assigned channel i if and only if $f(e(i)) > 0$. Note that for the flow $f(e(i))$ the implied channel assignment may not be feasible (it may require more than $I(v)$ channels at node v). The channel assignment algorithm transforms the given flow to fix this infeasibility. Below only a sketch of the algorithm is given. More details can be found in [1].

First observe that in an idle scenario, where all nodes v have the same number of interfaces I (i. e. $I = I(v)$) and where the number of available channels K is also I, the channel assignment implied by the LP (1) is feasible. This is because even the trivial channel assignment where all nodes are assigned all the channels 1 to I is feasible. The main idea behind the algorithm is to first transform the LP (1) solution to a new flow in which every edge e has flow $f(e(i)) > 0$ only for the channels $1 \le i \le I$. The basic operation that the algorithm uses for this is to equally distribute, for every edge e, the flow $f(e(i))$, for $I < i \le K$ to the edges $e(j)$, for $1 \le i \le I$. This ensures that all $f(e(i)) = 0$, for $I < i \le K$ after the operation. This operation is called Phase I of the Algorithm. Note that the Phase I operation does not violate the flow conservation constraints or the node radio constraints (5) in the LP (1). It can be shown that in the resulting solution the flow $f(e(i))$ may exceed the capacity of edge e by at most a factor $\phi = K/I$. This is called the "inflation factor" of Phase I. Likewise in the new flow, the *link congestion constraints* (5) may also be violated for edge e and channel i by no more than the inflation factor φ. In other words in the resulting flow

$$\frac{f(e(i))}{c(e)}+\sum_{e'\in I(e)}\frac{f(e'(i))}{c(e')}\le \phi c(q)\,.$$

This implies that if the new flow is scaled by a fraction $1/\phi$ than it is feasible for the LP (1). Note that the im-

plied channel assignment (assign channels 1 to I to every node) is also feasible. Thus, the above algorithm finds a feasible channel assignment with a λ value of at least λ^*/ϕ.

One shortcoming of the channel assignment algorithm (Phase I) described so far is that it only uses I of the K available channels. By using more channels the interference may be further reduced thus allowing for more flow to be pushed in the system. The channel assignment algorithm uses an additional heuristic for this improvement. This is called Phase II of the algorithm.

Now define an operation called "channel switch operation." Let A be a maximal connected component (the vertices in A are not connected to vertices outside A) in the graph formed by the edges e for a given channel i for which $f(e(i)) > 0$. The main observation to use is that for a given channel j, the operation of completely moving flow $f(e(i))$ to flow $f(e(j))$ for every edge e in A, does not impact the feasibility of the implied channel assignment. This is because there is no increase in the number of channels assigned per node after the flow transformation: the end nodes of edges e in A which were earlier assigned channel i are now assigned channel j instead. Thus, the transformation is equivalent to switching the channel assignment of nodes in A so that channel i is discarded and channel j is gained if not already assigned.

The Phase II heuristic attempts to re-transform the unscaled Phase I flows $f(e(i))$ so that there are multiple connected components in the graphs $G(e, i)$ formed by the edges e for each channel $1 \leq i \leq I$. This re-transformation is done so that the LP constraints are kept satisfied with an inflation factor of at most φ, as is the case for the unscaled flow after Phase I of the algorithm.

Next in Phase III of the algorithm the connected components within each graph $G(e, i)$ are grouped such that there are as close to K (but no more than) groups overall and such that the maximum interference within each group is minimized. Next the nodes within the lth group are assigned channel l, by using the channel switch operation to do the corresponding flow transformation. It can be shown that the channel assignment implied by the flow in Phase III is feasible. In addition the underlying flows $f(e(i))$ satisfy the LP (1) constraints with an inflation factor of at most $\phi = K/I$.

Next the algorithm scales the flow by the largest possible fraction (at least $1/\phi$) such that the resulting flow is a feasible solution to the LP (1) and also implies a feasible channel assignment solution to the channel assignment. Thus, the overall algorithm finds a feasible channel assignment (by not necessarily restricting to channels 1 to I only) with a λ value of at least λ^*/ϕ.

Link Flow Scheduling

The results in this section are obtained by extending those of [4] for the single channel case and for the Protocol Model of interference [2]. Recall that the time slotted schedule S is assumed to be periodic (with period T) where the indicator variable $X_{e,i,\tau}, e \in E, i \in F(e), \tau \geq 1$ is 1 if and only if link e is active in slot τ on channel i and i is a channel in common among the set of channels assigned to the end-nodes of edge e.

Directly applying the result (Claim 2) in [4] it follows that a necessary condition for interference free link scheduling is that for every $e \in E, i \in F(e), \tau \geq 1$: $X_{e,i,\tau} + \sum_{e' \in I(e)} X_{e',i,\tau} \leq c(q)$. Here $c(q)$ is a constant that only depends on the interference model. In the interference model this constant is a function of the fixed value q, the ratio of the interference range R_I to the transmission range R_T, and an intuition for its derivation for a particular value $q = 2$ is given below.

Lemma 1 *$c(q) = 8$ for $q = 2$.*

Proof Recall that an edge $e' \in I(e)$ if there exist two nodes $x, y \in V$ which are at most $2R_T$ apart and such that edge e is incident on node x and edge e' is incident on node y. Let $e = (u, v)$. Note that u and v are at most R_T apart. Consider the region C formed by the union of two circles C_u and C_v of radius $2R_T$ each, centered at node u and node v, respectively. Then $e' = (u', v') \in I(e)$ if an only if at least one of the two nodes u', v' is in C; Denote such a node by $C(e')$. Given two edges $e_1, e_2 \in I(e)$ that do not interfere with each other it must be the case that the nodes $C(e_1)$ and $C(e_2)$ are at least $2R_T$ apart. Thus, an upper bound on how many edges in $I(e)$ do not pair-wise interfere with each other can be obtained by computing how may nodes can be put in C that are pair-wise at least $2R_T$ apart. It can be shown [1] that this number is at most 8. Thus, in schedule S in a given slot only one of the two possibilities exist: either edge e is scheduled or an "independent" set of edges in $I(e)$ of size at most 8 is scheduled implying the claimed bound.□

A necessary condition: (*Link Congestion Constraint*) Recall that $\frac{1}{T}\sum_{1 \leq \tau \leq T} X_{e,i,\tau} = \frac{f(e(i))}{c(e)}$. Thus: Any valid "interference free" edge flows must satisfy for every link e and every channel i the Link Congestion Constraint:

$$\frac{f(e(i))}{c(e)} + \sum_{e' \in I(e)} \frac{f(e'(i))}{c(e')} \leq c(q). \tag{6}$$

A matching sufficient condition can also established [1].

A sufficient condition: (*Link Congestion Constraint*) If the edge flows satisfy for every link e and every channel i

the following *Link Schedulability Constraint* than an interference free edge communication schedule can be found using an algorithm given in [1].

$$\frac{f(e(i))}{c(e)} + \sum_{e' \in I(e)} \frac{f(e'(i))}{c(e')} \leq 1. \quad (7)$$

The above implies that if a flow $f(e(i))$ satisfies the *Link Congestion Constraint* then by scaling the flow by a fraction $1/c(q)$ it can be scheduled free of interference.

Key Results

Theorem *The RCL algorithm is a $Kc(q)/I$ approximation algorithm for the Joint Routing and Channel Assignment with Interference Free Edge Scheduling problem.*

Proof Note that the flow $f(e(i))$ returned by the channel assignment algorithm in Sect. "Channel Assignment" satisfies the *Link Congestion Constraint*. Thus, from the result of Sect. "Link Flow Scheduling" it follows that by scaling the flow by an additional factor of $1/c(q)$ the flow can be realized by an interference free link schedule. This implies a feasible solution to the joint routing, channel assignment and scheduling problem with a λ value of at least $\lambda^*/\phi c(q)$. Thus, the RCL algorithm is a $\phi c(q) = Kc(q)/I$ approximation algorithm. □

Applications

Infrastructure mesh networks are increasingly been deployed for commercial use and law enforcement. These deployment settings place stringent requirements on the performance of the underlying IWMNs. Bandwidth guarantee is one of the most important requirements of applications in these settings. For these IWMNs, topology change is infrequent and the variability of aggregate traffic demand from each mesh router (client traffic aggregation point) is small. These characteristics admit periodic optimization of the network which may be done by a system management software based on traffic demand estimation. This work can be directly applied to IWMNs. It can also be used as a benchmark to compare against heuristic algorithms in multi-hop wireless networks.

Open Problems

For future work, it will be interesting to investigate the problem when routing solutions can be enforced by changing link weights of a distributed routing protocol such as OSPF. Also, can the worst case bounds of the algorithm be improved (e. g. a constant factor independent of K and I)?

Cross References

- ▶ Graph Coloring
- ▶ Stochastic Scheduling

Recommended Reading

1. Alicherry, M., Bhatia, R., Li, L.E.: Joint channel assignment and routing for throughput optimization in multi-radio wireless mesh networks. In: Proc. ACM MOBICOM 2005, pp. 58–72
2. Gupta, P., Kumar, P.R.: The Capacity of Wireless Networks. IEEE Trans. Inf. Theory, IT-**46**(2), 388–404 (2000)
3. Jain, K., Padhye, J., Padmanabhan, V.N., Qiu, L.: Impact of interference on multi-hop wireless network performance. In: Proc. ACM MOBICOM 2003, pp. 66–80
4. Kumar, V.S.A., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: Algorithmic aspects of capacity in wireless networks. In: Proc. ACM SIGMETRICS 2005, pp. 133–144
5. Kumar, V.S.A., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: End-to-end packet-scheduling in wireless ad-hoc networks. In: Proc. ACM-SIAM symposium on Discrete algorithms 2004, pp. 1021–1030
6. Kyasanur, P., Vaidya, N.: Capacity of multi-channel wireless networks: Impact of number of channels and interfaces. In: Proc. ACM MOBICOM, pp. 43–57. 2005

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach

1994; Yang, Wong

Honghua Hannah Yang[1], Martin D. F. Wong[2]
[1] Strategic CAD Labs, Intel Corporation, Hillsboro, USA
[2] Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA

Keywords and Synonyms

Hypergraph partitioning; Netlist partitioning

Problem Definition

Circuit partitioning is a fundamental problem in many areas of VLSI layout and design. *Min-cut balanced bipartition* is the problem of partitioning a circuit into two disjoint components with equal weights such that the number of nets connecting the two components is minimized. The min-cut balanced bipartition problem was shown to be NP-complete [5]. The problem has been solved by heuristic algorithms, e. g., Kernighan and Lin type (K&L) iterative improvement methods [4,11], simulated annealing

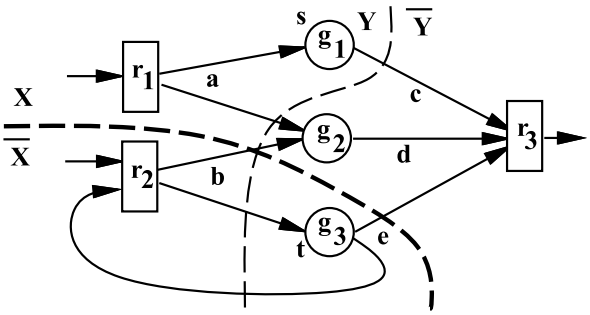
Algorithm: Flow-Balanced-Bipartition (FBB)
1. Pick a pair of nodes s and t in N;
2. Find a min-net-cut C in N;
   Let X be the subcircuit reachable from s through augmenting paths in the flow network, and $\bar{X}$ the rest;
3. **if** $(1-\epsilon)rW \leq w(X) \leq (1+\epsilon)rW$
   return C as the answer;
4. **if** $w(X) < (1-\epsilon)rW$
4.1. Collapse all nodes in X to s;
4.2. Pick a node $v \in \bar{X}$ adjacent to C and collapse it to s;
4.3. Goto 1;
5. **if** $w(X) > (1+\epsilon)rW$
5.1. Collapse all nodes in $\bar{X}$ to t;
5.2. Pick a node $v \in X$ adjacent to C and collapse it to t;
5.3. Goto 1;

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Figure 1
FBB algorithm

Procedure: Incremental Flow Computation
1. **while** ∃ an additional augmenting path from s to t
   increase flow value along the augmenting path;
2. Mark all nodes u s.t. ∃ an augmenting path from s to u;
3. Let C' be the set of bridging edges whose starting nodes are marked and ending nodes are not marked;
4. Return the nets corresponding to the bridging edges in C' as the min-net-cut C, and the marked nodes as X.

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Figure 2
Incremental max-flow computation

approaches [10], and analytical methods for the ratio-cut objective [2,7,13,15]. Although it is a natural method for finding a min-cut, the network max-flow min-cut technique [6,8] has been overlooked as a viable approach for circuit partitioning. In [16], a method was proposed for exactly modeling a circuit netlist (or, equivalently, a hypergraph) by a flow network, and an algorithm for balanced bipartition based on repeated applications of the max-flow min-cut technique was proposed as well. Our algorithm has the same asymptotic time complexity as one max-flow computation.



Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Figure 3
A circuit netlist with two net-cuts

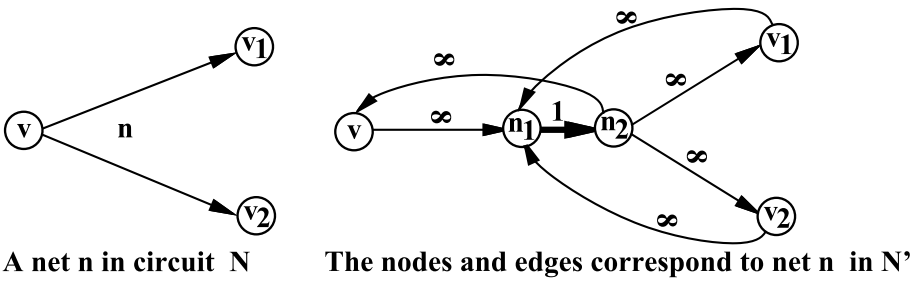
A *circuit netlist* is defined as a digraph $N = (V, E)$, where V is a set of nodes representing logic gates and registers and E is a set of edges representing wires between gates and registers. Each node $v \in V$ has a weight $w(v) \in R^+$. The total weight of a subset $U \subseteq V$ is denoted by $w(U) = \Sigma_{v \in U} w(v)$. $W = w(V)$ denotes the total weight of the circuit. A *net* $n = (v; v_1, \ldots, v_l)$ is a set of outgoing edges from node v in N. Given two nodes s and t in N, an $s-t$ *cut* (or *cut* for short) $(X, \bar{X})$ of N is a bipartition of the nodes in V such that $s \in X$ and $t \in \bar{X}$. The *net-cut* $net(X, \bar{X})$ of the cut is the set of nets in N that are incident to nodes in both X and $\bar{X}$. A cut $(X, \bar{X})$ is a *min-net-cut* if $|net(X, \bar{X})|$ is minimum among all $s-t$ cuts of N. In Fig. 3, net $a = (r_1; g_1, g_2)$, net cuts $net(X, \bar{X}) = \{b, e\}$ and $net(Y, \bar{Y}) = \{c, a, b, e\}$, and $(X, \bar{X})$ is a min-net-cut.

Formally, given an aspect ratio r and a deviation factor ϵ, *min-cut r-balanced bipartition* is the problem of finding a bipartition $(X, \bar{X})$ of the netlist N such that (1) $(1-\epsilon)rW \leq W(X) \leq (1+\epsilon)rW$ and (2) the size of the cut $net(X, \bar{X})$ is minimum among all bipartitions satisfying (1). When $r = 1/2$, this becomes a min-cut balanced-bipartition problem.

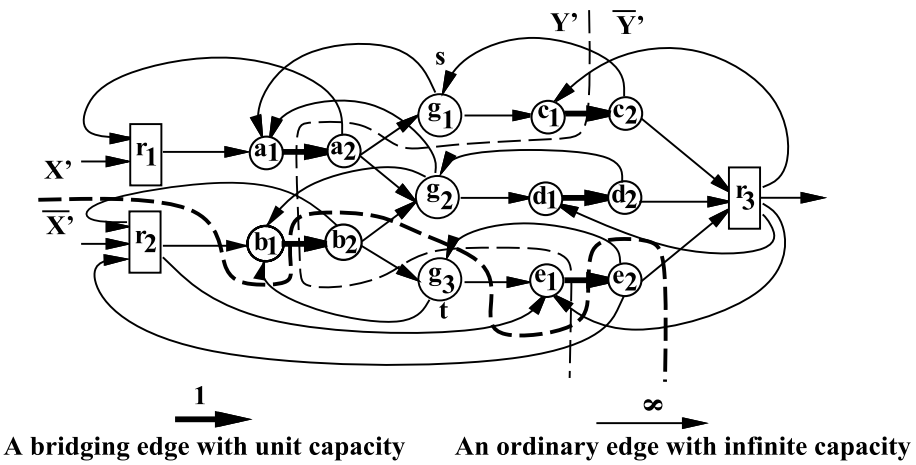
Key Results

Optimal-Network-Flow-Based Min-Net-Cut Bipartition

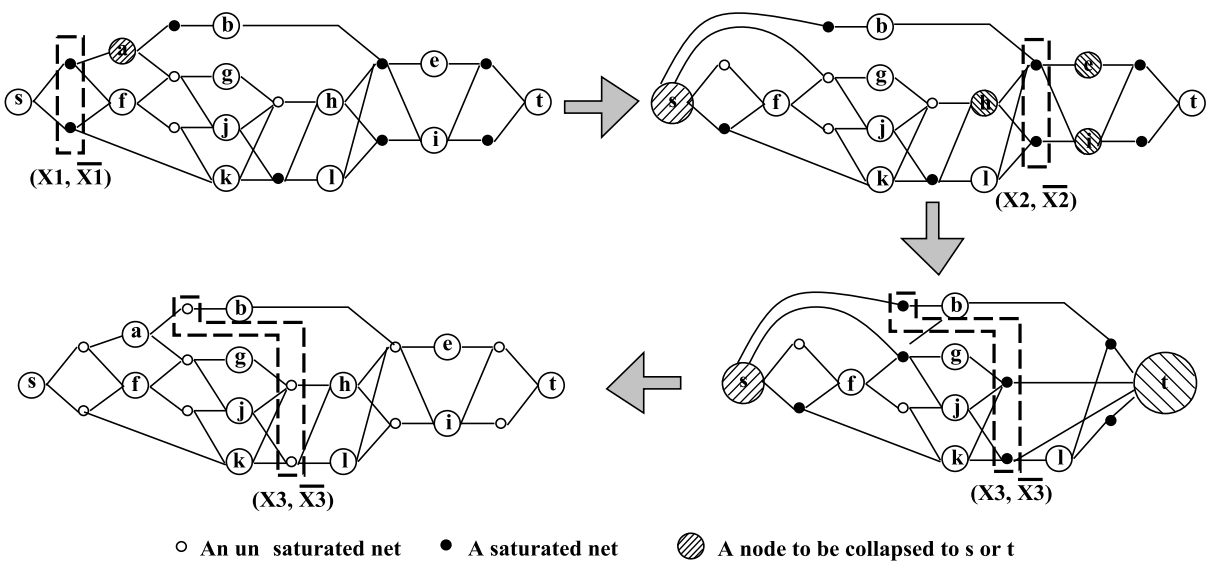
The problem of finding a min-net-cut in $N = (V, E)$ is reduced to the problem of finding a cut of minimum capacity. Then the latter problem is solved using the max-flow min-cut technique. A flow network $N' = (V', E')$ is constructed from $N = (V, E)$ as follows (see Figs. 4 and 5):
1. V' contains all nodes in V.
2. For each net $n = (v; v_1, \ldots, v_l)$ in N, add two nodes n_1 and n_2 in V' and a *bridging edge* $bridge(n) = (n_1, n_2)$ in E'.



Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Figure 4
Modeling a net in *N* in the flow network *N'*



Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Figure 5
The flow network for Fig. 3



Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Figure 6
FBB on the example in Fig. 5 for $r = 1/2$, $\epsilon = 0.15$ and unit weight for each node. The algorithm terminates after finding cut $(X_2, \bar{X}_2)$. A small solid node indicates that the bridging edge corresponding to the net is saturated with flow

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Table 1
Comparison of SN, PFM3, and FBB ($r = 1/2, \epsilon = 0.1$)

Circuit				Avg. net-cut size			FBB bipart. ratio	Improve. %	
Name	Gates and latches	Nets	Avg. deg	SN	PFM3	FBB		Over SN	Over PFM3
C1355	514	523	3.0	38.9	29.1	26.0	1:1.08	33.2	10.7
C2670	1161	1254	2.6	51.9	46.0	37.1	1:1.15	28.5	19.3
C3540	1667	1695	2.7	90.3	71.0	79.8	1:1.11	11.6	−12.4
C7552	3466	3565	2.7	44.3	81.8	42.9	1:1.08	3.2	47.6
S838	478	511	2.6	27.1	21.0	14.7	1:1.04	45.8	30.0
Ave							1:1.10	24.5	19.0

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Table 2
Comparison of EIG1, PB, and FBB ($r = 1/2, \epsilon = 0.1$). All allow $\preceq$ 10% deviation

Circuit				Best net-cut size			Improve. % over		FBB elaps. sec.
Name	Gates and latches	Nets	Avg. deg	EIG1	PB	FBB	EIG1	PB	
S1423	731	743	2.7	23	16	13	43.5	18.8	1.7
S9234	5808	5805	2.4	227	74	70	69.2	5.4	55.7
S13207	8696	8606	2.4	241	91	74	69.3	18.9	100.0
S15850	10310	10310	2.4	215	91	67	68.8	26.4	96.5
S35932	18081	17796	2.7	105	62	49	53.3	21.0	2808
S38584	20859	20593	2.7	76	55	47	38.2	14.5	1130
S38417	24033	23955	2.4	121	49	58	52.1	−18.4	2736
Average							58.5	11.3	

3. For each node $u \in \{v, v_1, \ldots, v_l\}$ incident on net n, add two edges (u, n_1) and (n_2, u) in E'.
4. Let s be the source of N' and t the sink of N'.
5. Assign unit capacity to all bridging edges and infinite capacity to all other edges in E'.
6. For a node $v \in V'$ corresponding to a node in V, $w(v)$ is the weight of v in N. For a node $u \in V'$ split from a net, $w(u) = 0$.

Note that all nodes incident on net n are connected to n_1 and are connected from n_2 in N'. Hence the flow network construction is symmetric with respect to all nodes incident on a net. This construction also works when the netlist is represented as a hypergraph.

It is clear that N' is a strongly connected digraph. This property is the key to reducing the bidirectional min-net-cut problem to a minimum-capacity cut problem that counts the capacity of the forward edges only.

Theorem 1 *N has a cut of net-cut size at most C if and only if N' has a cut of capacity at most C.*

Corollary 1 *Let $(X', \bar{X}')$ be a cut of minimum capacity C in N'. Let $N_{cut} = \{n \mid bridge(n) \in (X', \bar{X}')\}$. Then $N_{cut} = (X, \bar{X})$ is a min-net-cut in N and $|N_{cut}| = C$.*

Corollary 2 *A min-net-cut in a circuit $N = (V, E)$ can be found in $O(|V||E|)$ time.*

Min-Cut Balanced-Bipartition Heuristic

First, a repeated max-flow min-cut heuristic algorithm, flow-balanced bipartition (FBB), is developed for finding an r-balanced bipartition that minimizes the number of crossing nets. Then, an efficient implementation of FBB is developed that has the same asymptotic time complexity as one max-flow computation. For ease of presentation, the FBB algorithm is described on the original circuit rather than the flow network constructed from the circuit. The heuristic algorithm is described in Fig. 1. Figure 6 shows an example.

Table 2 compares the best bipartition net-cut sizes of FBB with those produced by the analytical-method-based partitioners EIG1 (Hagen and Kahng [7]) and PARABOLI (PB) (Riess et al. [13]). The results produced by PARABOLI were the best previously known results reported on the benchmark circuits. The results for FBB were the best of ten runs. On average, FBB outperformed EIG1 and PARABOLI by 58.1% and 11.3% respectively. For circuit S38417, the suboptimal result from FBB can be improved by (1) running more times and (2) applying clustering techniques to the circuit based on connectivity before partitioning.

In the FBB algorithm, the node-collapsing method is chosen instead of a more gradual method (e. g., [9]) to en-

sure that the capacity of a cut always reflects the real net-cut size. To pick a node at steps 4.2 and 5.2, a threshold R is given for the number of nodes in the uncollapsed subcircuit. A node is randomly picked if the number of nodes is larger than R. Otherwise, all nodes adjacent to C are tried and the one whose collapse induces a min-net-cut with the smallest size is picked. A naive implementation of step 2 by computing the max-flow from the zero flow would incur a high time complexity. Instead, the flow value in the flow network is retained, and additional flow is explored to saturate the bridging edges of the min-net-cut from one iteration to the next. The procedure is shown in Fig. 2. Initially, the flow network retains the flow function computed in the previous iteration. Since the max-flow computation using the augmenting-path method is insensitive to the initial flow values in the flow network and the order in which the augmenting paths are found, the above procedure correctly finds a max-flow with the same flow value as a max-flow computed in the collapsed flow network from the zero flow.

Theorem 2 *FBB has time complexity $O(|V||E|)$ for a connected circuit $N = (V, E)$.*

Theorem 3 *The number of iterations and the final net-cut size are nonincreasing functions of ϵ.*

In practice, FBB terminates much faster than this worst-case time complexity as shown in the Sect. "Experimental Results". Theorem 3 allows us to improve the efficiency of FBB and the partition quality for a larger ϵ. This is not true for other partitioning approaches such as the K&L heuristics.

Applications

Circuit partitioning is a fundamental problem in many areas of VLSI layout and design automation. The FBB algorithm provides the first efficient predictable solution to the min-cut balanced-circuit-partitioning problem. It directly relates the efficiency and the quality of the solution produced by the algorithm to the deviation factor ϵ. The algorithm can be easily extended to handle nets with different weights by simply assigning the weight of a net to its bridging edge in the flow network. K-way min-cut partitioning for $K > 2$ can be accomplished by recursively applying FBB or by setting $r = 1/K$ and then using FBB to find one partition at a time. A flow-based method for directly solving the problem can be found in [12]. Prepartitioning circuit clustering according to the connectivity or the timing information of the circuit can be easily incorporated into FBB by treating a cluster as a node. Heuristic solutions based on K&L heuristics or simulated annealing with low temperature can be used to further fine-tune the solution.

Experimental Results

The FBB algorithm was implemented in SIS/MISII [1] and tested on a set of large ISCAS and MCNC benchmark circuits on a SPARC 10 workstation with 36-MHz CPU and 32 MB memory.

Table 1 compares the average bipartition results of FBB with those reported by Dasdan and Aykanat in [3]. SN is based on the K&L heuristic algorithm in Sanchis [14]. PFM3 is based on the K&L heuristic with free moves as described in [3]. For each circuit, SN was run 20 times and PFM3 10 times from different randomly generated initial partitions. FBB was run 10 times from different randomly selected s and t. With only one exception, FBB outperformed both SN and PFM3 on the five circuits. On average, FBB found a bipartition with 24.5% and 19.0% fewer crossing nets than SN and PFM3 respectively. The runtimes of SN, PFM3, and FBB were not compared since they were run on different workstations.

Cross References

- ▶ Approximate Maximum Flow Construction
- ▶ Circuit Placement
- ▶ Circuit Retiming
- ▶ Max Cut
- ▶ Minimum Bisection
- ▶ Multiway Cut
- ▶ Separators in Graphs

Recommended Reading

1. Brayton, R.K., Rudell, R., Sangiovanni-Vincentelli, A.L.: MIS: A Multiple-Level Logic Optimization. IEEE Trans. CAD **6**(6), 1061–1081 (1987)
2. Cong, J., Hagen, L., Kahng, A.: Net Partitions Yield Better Module Partitions. In: Proc. 29th ACM/IEEE Design Automation Conf., 1992, pp. 47–52
3. Dasdan, A., Aykanat, C.: Improved Multiple-Way Circuit Partitioning Algorithms. In: Int. ACM/SIGDA Workshop on Field Programmable Gate Arrays, Feb. 1994
4. Fiduccia, C.M., Mattheyses, R.M.: A Linear Time Heuristic for Improving Network Partitions. In: Proc. ACM/IEEE Design Automation Conf., 1982, pp. 175–181
5. Garey, M., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, Gordonsville (1979)
6. Goldberg, A.W., Tarjan, R.E.: A New Approach to the Maximum Flow Problem. J. SIAM **35**, 921–940 (1988)

7. Hagen, L., Kahng, A.B.: Fast Spectral Methods for Ratio Cut Partitioning and Clustering. In: Proc. IEEE Int. Conf. on Computer-Aided Design, November 1991, pp. 10–13
8. Hu, T.C., Moerder, K.: Multiterminal Flows in a Hypergraph. In: Hu, T.C., Kuh, E.S. (eds.) VLSI Circuit Layout: Theory and Design, pp. 87–93. IEEE Press (1985)
9. Iman, S., Pedram, M., Fabian, C., Cong, J.: Finding Uni-Directional Cuts Based on Physical Partitioning and Logic Restructuring. In: 4th ACM/SIGDA Physical Design Workshop, April 1993
10. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Science **4598**, 671–680 (1983)
11. Kernighan, B., Lin, S.: An Efficient Heuristic Procedure for Partitioning of Electrical Circuits. Bell Syst. Tech. J., 291–307 (1970)
12. Liu, H., Wong, D.F.: Network-Flow-based Multiway Partitioning with Area and Pin Constraints. IEEE Trans. CAD Integr. Circuits Syst. **17**(1), 50–59 (1998)
13. Riess, B.M., Doll, K., Frank, M.J.: Partitioning Very Large Circuits Using Analytical Placement Techniques. In: Proc. 31th ACM/IEEE Design Automation Conf., 1994, pp. 646–651
14. Sanchis, L.A.: Multiway Network Partitioning. IEEE Trans. Comput. **38**(1), 62–81 (1989)
15. Wei, Y.C., Cheng, C.K.: Towards Efficient Hierarchical Designs by Ratio Cut Partitioning. In: Proc. IEEE Int. Conf. on Computer-Aided Design, November 1989, pp. 298–301
16. Yang, H., Wong, D.F.: Efficient Network Flow Based Min-Cut Balanced Partitioning. In: Proc. IEEE Int. Conf. on Computer-Aided Design, 1994, pp. 50–55

Circuit Placement

2000; Caldwell, Kahng, Markov
2002; Kennings, Markov
2006; Kennings, Vorwerk

ANDREW A. KENNINGS[1], IGOR L. MARKOV[2]
[1] Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada
[2] Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA

Keywords and Synonyms

EDA; Netlist; Layout; Min-cut placement; Min-cost max-flow; Analytical placement; Mathematical programming

Problem Definition

This problem is concerned with efficiently determining constrained positions of objects while minimizing a measure of interconnect between the objects, as in physical layout of integrated circuits, commonly done in 2-dimensions. While most formulations are NP-hard, modern circuits are so large that practical algorithms for placement must have near-linear runtime and memory requirements, but not necessarily produce optimal solutions. While early software for circuit placement was based on Simulated Annealing, research in algorithms identified more scalable techniques which are now being adopted in the Electronic Design Automation industry.

One models a circuit by a hypergraph $G_h(V_h,E_h)$ with (i) vertices $V_h = \{v_1, \ldots, v_n\}$ representing logic gates, standard cells, larger modules, or fixed I/O pads and (ii) hyperedges $E_h = \{e_1, \ldots, e_m\}$ representing connections between modules. Every incident pair of a vertex and a hyperedge connect through a *pin* for a total of P pins in the hypergraph. Each vertex $v_i \in V_h$ has width w_i, height h_i and area A_i. Hyperedges may also be weighted. Given G_h, circuit placement seeks center positions (x_i,y_i) for vertices that optimize a hypergraph-based objective subject to constraints (see below). A placement is captured by $\mathbf{x} = (x_1, \cdots, x_n)$ and $\mathbf{y} = (y_1, \cdots, y_n)$.

Objective Let C_k be the index set of the hypergraph vertices incident to hyperedge e_k. The total half-perimeter wirelength (HPWL) of the circuit hypergraph is given by $\text{HPWL}(G_h) = \sum_{e_k \in E_h} \text{HPWL}(e_k) = \sum_{e_k \in E_h} \left[\max_{i,j \in C_k} |x_i - x_j| + \max_{i,j \in C_k} |y_i - y_j|\right]$. HPWL is piece-wise linear, separable in the x and y directions, convex, but not strictly convex. Among many objectives for circuit placement, it is the simplest and most common.

Constraints

1. **No overlap.** The area occupied by any two vertices cannot overlap; i. e., either $|x_i - x_j| \geq \frac{1}{2}(w_i + w_j)$ or $|y_i - y_j| \geq \frac{1}{2}(h_i + h_j)$, $\forall v_i, v_j \in V_h$.
2. **Fixed outline.** Each vertex $v_i \in V_h$ must be placed entirely within a specified rectangular region bounded by $x_{\min}(y_{\min})$ and $x_{\max}(y_{\max})$ which denote the left (bottom) and right (top) boundaries of the specified region.
3. **Discrete slots.** There is only a finite number of discrete positions, typically on a grid. However, in large-scale circuit layout, slot constraints are often ignored during *global placement*, and enforced only during *legalization* and *detail placement*.

Other constraints may include alignment, minimum and maximum spacing, etc. Many placement techniques temporarily relax overlap constraints into density constraints to avoid vertices clustered in small regions. A $m \times n$ regular bin structure B is superimposed over the fixed outline and vertex area is assigned to bins based on the positions of vertices. Let D_{ij} denote the density of bin $B_{ij} \in B$, defined as the total cell area assigned to bin B_{ij} divided by its capacity. Vertex overlap is limited implicitly by $D_{ij} \leq K$, $\forall B_{ij} \in B$, for some $K \leq 1$ (density target).

Problem 1 (Circuit Placement)
INPUT: *Circuit hypergraph $G_h(V_h,E_h)$ and a fixed outline for the placement area.*
OUTPUT: *Positions for each vertex $v_i \in V_h$ such that (1) wirelength is minimized and (2) the area-density constraints $D_{ij} \leq K$ are satisfied for all $B_{ij} \in B$.*

Key Results

An unconstrained optimal position of a single placeable vertex connected to fixed vertices can be found in linear time as the median of adjacent positions [8]. Unconstrained HPWL minimization for multiple placeable vertices can be formulated as a linear program [7,10]. For each $e_k \in E_h$, upper and lower bound variables U_k and L_k are added. The cost of e_k (x-direction only) is the difference between U_k and L_k. Each $U_k(L_k)$ comes with p_k inequality constraints that restricts its value to be larger (smaller) than the position of every vertex $i \in C_k$. A hypergraph with n vertices and m hyperedges is represented by a linear program with $n + 2m$ variables and $2P$ constraints.

Linear programming has poor scalability, and integrating constraint-tracking into optimization is difficult. Other approaches include non-linear optimization and partitioning-based methods.

Combinatorial Techniques for Wirelength Minimization

The no-overlap constraints are not convex and cannot be directly added to the linear program for HPWL minimization. Such a program is first solved directly or by casting its dual as an instance of the min-cost max-flow problem [12]. Vertices often cluster in small regions of high density. One can lower-bound the distance between closely-placed vertices with a single linear constraint that depends on the relative placement of these vertices [10]. The resulting optimization problem is incrementally re-solved, and the process repeats until the desired density is achieved.

The *min-cut placement* technique is based on balanced min-cut partitioning of hypergraphs and is more focused on density constraints [11]. Vertices of the initial hypergraph are first partitioned in two similar-sized groups. One of them is assigned to the left half of the placement region, and the other one to the right half. Partitioning is performed by the Multi-level Fiduccia–Mattheyses (MLFM) heuristic [9] to minimize connections between the two groups of vertices (the net-cut objective). Each half is partitioned again, but takes into account the connections to the other half [11]. At the large scale, ensuring the similar sizes of bi-partitions corresponds to density constraints and cut minimization corresponds to HPWL minimization. When regions become small and contain < 10 vertices, optimal positions can be found with respect to discrete slot constraints by branch-and-bound [2]. Balanced hypergraph partitioning is NP-hard [4], but the MLFM heuristic takes $O((V + E)\log V)$ time. The entire min-cut placement procedure takes $O((V + E)(\log V)^2)$ time and can process hypergraphs with millions of vertices in several hours.

A special case of interest is that of one-dimensional placement. When all vertices have identical width and none of them are fixed, one obtains the NP-hard MINIMUM LINEAR ARRANGEMENT problem [4] which can be approximated in polynomial time within $O(\log V)$ and solved exactly for trees in $O(V^3)$ time as shown by Yannakakis. The min-cut technique described above also works well for the related NP-hard MINIMUM-CUT LINEAR ARRANGEMENT problem [4].

Nonlinear Optimization

Quadratic and generic non-linear optimization may be faster than linear programming, while reasonably approximating the original formulation. The hypergraph is represented by a weighted graph where w_{ij} represents the weight on the 2-pin edge connecting vertices v_i and v_j in the weighted graph. When an edge is absent, $w_{ij} = 0$, and in general $w_{ii} = -\Sigma_{i \neq j} w_{ij}$.

Quadratic Placement A quadratic placement (x-direction only) is given by

$$\Phi(x) = \sum_{i,j} w_{ij}\left[(x_i - x_j)^2\right] = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{c}^{\mathrm{T}}\mathbf{x} + \text{const.} \quad (1)$$

The global minimum of $\Phi(x)$ is found by solving $\mathbf{Qx}+\mathbf{c} = \mathbf{0}$ which is a sparse, symmetric, positive-definite system of linear equations (assuming ≥ 1 fixed vertex), efficiently solved to sufficient accuracy using any number of iterative solvers. Quadratic placement may have different optima depending on the model (clique or star) used to represent hyperedges. However, for a k-pin hyperedge, if the weight on the 2-pin edges introduced is set to W_c in the clique mode and kW_c in the star model, then the models are equivalent in quadratic placement [7].

Linearized Quadratic Placement Quadratic placement can produce lower quality placements. To approximate the linear objective, one can iteratively solve Eq. (1) with $w_{ij} = 1/|x_i - x_j|$ computed at every iteration. Alternatively, one can solve a single β-regularized optimization problem given by $\Phi^{\beta}(\mathbf{x}) = \min_x \sum_{i,j} w_{ij}\sqrt{(x_i - x_j)^2 + \beta}$, $\beta > 0$,

e. g., using a Primal-Dual Newton method with quadratic convergence [1].

Half-Perimeter Wirelength Placement HPWL can be provably approximated by strictly convex and differentiable functions. For 2-pin hyperedges, β-regularization can be used [1]. For an m-pin hyperedge ($m \geq 3$), one can rewrite HPWL as the maximum (l_∞-norm) of all $m(m-1)/2$ pairwise distances $|x_i - x_j|$ and approximate the l_∞-norm by the l_p-norm (p-th root of the sum of p-th powers). This removes all non-differentiabilities except at $\mathbf{0}$ which is then removed with β-regularization. The resulting HPWL approximation is given by

$$\mathrm{HPWL}_{p-\beta-\mathrm{reg}}(G_h) = \sum_{e_k \in E_h} \left(\sum_{i,j \in C_k} |x_i - x_j|^p + \beta \right)^{1/p} \tag{2}$$

which overestimates HPWL with arbitrarily small relative error as $p \to \infty$ and $\beta \to 0$ [7]. Alternatively, HPWL can be approximated via the log-sum-exp formula given by

$$\mathrm{HPWL}_{\text{log-sum-exp}}(G_h) = \alpha \sum_{e_k \in E_h} \left[\ln \left(\sum_{i \in C_k} \exp\left(\frac{x_i}{\alpha}\right) \right) + \ln \left(\sum_{v_i \in C_k} \exp\left(\frac{-x_i}{\alpha}\right) \right) \right] \tag{3}$$

where $\alpha > 0$ is a smoothing parameter [6]. Both approximations can be optimized using conjugate gradient methods.

Analytic Techniques for Target Density Constraints

The target density constraints are non-differentiable and are typically handled by approximation.

Force-Based Spreading The key idea is to add constant forces $\mathbf{f}$ that pull vertices always from overlaps, and recompute the forces over multiple iterations to reflect changes in vertex distribution. For quadratic placement, the new optimality conditions are $\mathbf{Qx} + \mathbf{c} + \mathbf{f} = \mathbf{0}$ [8]. The constant force can perturb a placement in any number of ways to satisfy the target density constraints. The force $\mathbf{f}$ is computed using a discrete version of Poisson's equation.

Fixed-Point Spreading A fixed point f is a pseudo-vertex with zero area, fixed at (x_f, y_f), and connected to one vertex $H(f)$ in the hypergraph through the use of a pseudo-edge with weight $w_{f,H(f)}$. Quadratic placement with fixed points is given by $\Phi(x) = \sum_{i,j} w_{i,j}(x_i - x_j)^2 + \sum_f w_{f,H(f)}(x_{H(f)} - x_f)^2$. Each each fixed point f introduces a quadratic term $w_{f,H(f)}(x_{H(f)} - x_f)^2$. By manipulating the positions of fixed points, one can perturb a placement to satisfy the target density constraints. Compared to constant forces, fixed points improve the controllability and stability of placement iterations [5].

Generalized Force-Directed Spreading The Helmholtz equation models a diffusion process and makes it ideal for spreading vertices [3]. The Helmholtz equation is given by

$$\frac{\partial^2 \phi(x,y)}{\partial x^2} + \frac{\partial^2 \phi(x,y)}{\partial y^2} - \epsilon \phi(x,y) = D(x,y)\,, \quad (x,y) \in R \frac{\partial \phi}{\partial \nu} = 0\,, \quad (x,y) \text{ on the boundary of } R \tag{4}$$

where $\epsilon > 0$, ν is an outer unit normal, R represents the fixed outline, and $D(x,y)$ represents the continuous density function. The boundary conditions, $\partial\phi/\partial\nu = 0$, specify that forces pointing outside of the fixed outline be set to zero – this is a key difference with the Poisson method which assumes that forces become zero at infinity. The value ϕ_{ij} at the center of each bin B_{ij} is found by discretization of Eq. (4) using finite differences. The density constraints are replaced by $\phi_{ij} = \hat{K}, \forall B_{ij} \in B$ where $\hat{K}$ is a scaled representative of the density target K. Wirelength minimization subject to the smoothed density constraints can be solved via Uzawa's algorithm. For quadratic wirelength, this algorithm is a generalization of force-based spreading.

Potential Function Spreading Target density constraints can also be satisfied via a penalty function. The area assigned to bin B_{ij} by vertex v_i is represented by Potential(v_i, B_{ij}) which is a bell-shaped function. The use of piecewise quadratic functions make the potential function non-convex, but smooth and differentiable [6]. The penalty term given by

$$\text{Penalty} = \sum_{B_{ij} \in B} \left(\sum_{v_i \in V_h} \text{Potential}(v_i, B_{ij}) - K \right)^2 \tag{5}$$

can be combined with a wirelength approximation to arrive at an unconstrained optimization problem which is solved using an efficient conjugate gradient method [6].

Applications

Practical applications involve more sophisticated interconnect objectives, such as circuit delay, routing congestion, power dissipation, power density, and maximum

thermal gradient. The above techniques are adapted to handle multi-objective optimization. Many such extensions are based on heuristic assignment of net weights that encourage the shortening of some (e. g., timing-critical and frequently-switching) connections at the expense of other connections. To moderate routing congestion, predictive congestion maps are used to decrease the maximal density constraint for placement in congested regions. Another application is in physical synthesis, where incremental placement is used to evaluate changes in circuit topology.

Experimental Results

Circuit placement has been actively studied for the past 30 years and a wealth of experimental results are reported throughout the literature. A 2003 result demonstrated that placement tools could produce results as much as 1.41× to 2.09× known optimal wirelengths on average (advances have been made since this study). A 2005 placement contest found that a set of tools produced placements with wirelengths that differed by as much as 1.84× on average. A 2006 placement contest found that a set of tools produced placements that differed by as much as 1.39× on average when the objective was the simultaneous minimization of wirelength, routability and run time. Placement run times range from minutes for smaller instances to hours for larger instances, with several millions of variables.

Data Sets

Benchmarks include the ICCAD '04 suite (http://vlsicad.eecs.umich.edu/BK/ICCAD04bench/), the ISPD '05 suite (http://www.sigda.org/ispd2005/contest.htm) and the ISPD '06 suite (http://www.sigda.org/ispd2006/contest.htm). Instances in these benchmark suites contain between 10K to 2.5M placeable objects. Other common suites can be found, including large-scale placement instances problems with known optimal solutions (http://cadlab.cs.ucla.edu/~pubbench).

Cross References

► Performance-Driven Clustering

Recommended Reading

1. Alpert, C.J., Chan, T., Kahng, A.B., Markov, I.L., Mulet, P.: Faster minimization of linear wirelength for global placement. IEEE Trans. CAD **17**(1), 3–13 (1998)
2. Caldwell, A.E., Kahng, A.B., Markov, I.L.: Optimal partitioners and end-case placers for standard-cell layout. IEEE Trans. CAD **19**(11), 1304–1314 (2000)
3. Chan, T., Cong, J., Sze, K.: Multilevel generalized force-directed method for circuit placement. Proc. Intl. Symp. Physical Design. ACM Press, San Francisco, 3–5 Apr 2005. pp. 185–192 (2005)
4. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and Approximation: Combinatorial optimization problems and their approximability properties. Springer (1998)
5. Hu, B., Marek-Sadowska, M.: Multilevel fixed-point-addition-based VLSI placement. IEEE Trans. CAD **24**(8), 1188–1203 (2005)
6. Kahng, A.B., Wang, Q.: Implementation and extensibility of an analytic placer. IEEE Trans. CAD **24**(5), 734–747 (2005)
7. Kennings, A., Markov, I.L.: Smoothing max-terms and analytical minimization of half-perimeter wirelength. VLSI Design **14**(3), 229–237 (2002)
8. Kennings, A., Vorwerk, K.: Force-directed methods for generic placement. IEEE Trans. CAD **25**(10), 2076–2087 (2006)
9. Papa, D.A., Markov, I.L.: Hypergraph partitioning and clustering. In: Gonzalez, T. (ed.) Handbook of algorithms. Taylor & Francis Group, Boca Raton, CRC Press, pp. 61–1 (2007)
10. Reda, S., Chowdhary, A.: Effective linear programming based placement methods. In: ACM Press, San Jose, 9–12 Apr 2006
11. Roy, J.A., Adya, S.N., Papa, D.A., Markov, I.L.: Min-cut floorplacement. IEEE Trans. CAD **25**(7), 1313–1326 (2006)
12. Tang, X., Tian, R., Wong, M.D.F.: Optimal redistribution of white space for wirelength minimization. In: Tang, T.-A. (ed.) Proc. Asia South Pac. Design Autom. Conf., ACM Press, 18–21 Jan 2005, Shanghai. pp. 412–417 (2005)

Circuit Retiming

1991; Leiserson, Saxe

Hai Zhou
Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA

Keywords and Synonyms

Min-period retiming; Min-area retiming

Problem Definition

Circuit retiming is one of the most effective structural optimization techniques for sequential circuits. It moves the registers within a circuit without changing its function. Besides clock period, retiming can be used to minimize the number of registers in the circuit. It is also called minimum area retiming problem. Leiserson and Saxe [3] started the research on retiming and proposed algorithms for both minimum period and minimum area retiming. Both their algorithms for minimum area and minimum period will be presented here.

The problems can be formally described as follows. Given a directed graph $G = (V, E)$ representing a circuit—each node $v \in V$ represents a gate and each edge $e \in E$ represents a signal passing from one gate to another—with gate delays $d\colon V \to \mathbb{R}^+$ and register numbers $w\colon E \to \mathbb{N}$, the minimum area problem asks for a relocation of registers $w'\colon E \to \mathbb{N}$ such that the number of registers in the circuit is minimum under a given clock period φ. The minimum period problem asks for a solution with the minimum clock period.

Notations

To guarantee that the new registers are actually a relocation of the old ones, a label $r\colon V \to \mathbb{Z}$ is used to represent how many registers are moved from the outgoing edges to the incoming edges of each node. Using this notation, the new number of registers on an edge (u, v) can be computed as

$$w'[u, v] = w[u, v] + r[v] - r[u] .$$

The same notation can be extended from edges to paths. However, between any two nodes u and v, there may be more than one path. Among these paths, the ones with the minimum number of registers will decide how many registers can be moved outside of u and v. The number is denoted by $W[u, v]$ for any $u, v \in V$, that is,

$$W[u, v] \triangleq \min_{p\colon u \rightsquigarrow v} \sum_{(x,y)\in p} w[x, y]$$

The maximal delay among all the paths from u to v with the minimum number of registers is also denoted by $D[u, v]$, that is,

$$D[u, v] \triangleq \max_{w[p\colon u \rightsquigarrow v]=W[u,v]} \sum_{x\in p} d[x]$$

Constraints

Based on the notations, a valid retiming r should not have any negative number of registers on any edge. Such a validity condition is given as

$$P0(r) \triangleq \forall (u, v) \in E\colon w[u, v] + r[v] - r[u] \geq 0$$

On the other hand, given a retiming r, the minimum number of registers between any two nodes u and v is $W[u, v] - r[u] + r[v]$. This number will not be negative because of the previous constraint. However, when it is zero, there will be a path of delay $D[u, v]$ without any register on it. Therefore, to have a retimed circuit working for clock period φ, the following constraint must be satisfied.

$$P1(r) \triangleq \forall u, v \in V\colon D[u, v] > \phi \Rightarrow W[u, v] + r[v] - r[u] \geq 1$$

Key Results

The object of the minimum area retiming is to minimize the total number of registers in the circuit, which is given by $\sum_{(u,v)\in E} w'[u, v]$. Expressing $w'[u, v]$ in terms of r, the objective becomes

$$\sum_{v\in V} (in[v] - out[v]) * r[v] + \sum_{(u,v)\in E} w[u, v]$$

where $in[v]$ is the in-degree and $out[v]$ is the out-degree of node v. Since the second term is a constant, the problem can be formulated as the following integer linear program.

$$\begin{aligned} &\text{Minimize} \sum_{v\in V} (in[v] - out[v]) * r[v] \\ &s.t.\ \ w[u, v] + r[v] - r[u] \geq 0 \quad \forall (u, v) \in E \\ &\qquad W[u, v] + r[v] - r[u] \geq 1 \quad \forall u, v \in V\colon D[u, v] > \phi \\ &\qquad r[v] \in \mathbb{Z} \quad \forall v \in V \end{aligned}$$

Since the constraints have only difference inequalities with integer constant terms, solving the relaxed linear program (without the integer constraint) will only give integer solutions. Even better, it can be shown that the problem is the dual of a minimum cost network flow problem, and thus can be solved efficiently.

Theorem 1 *The integer linear program for the minimum area retiming problem is the dual of the following minimum cost network flow problem.*

$$\begin{aligned} &\textit{Minimize} \sum_{(u,v)\in E} w[u, v] * f[u, v] \\ &\qquad + \sum_{D[u,v]>\phi} (W[u, v] - 1) * f[u, v] \\ &s.t.\ \ in[v] + \sum_{(v,w)\in E \vee D[v,w]>\phi} f[v, w] = out[v] \\ &\qquad + \sum_{(u,v)\in ED[u,v]>\phi} f[u, v] \quad \forall v \in V \\ &\qquad f[u, v] \geq 0 \quad \forall (u, v) \in ED[u, v] > \phi \end{aligned}$$

From the theorem, it can be seen that the network graph is a dense graph where a new edge (u, v) needs to be introduced for any node pair u, v such that $D[u, v] > \phi$.

There may be redundant constraints in the system. For example, if $W[u,w] = W[u,v] + w[v,w]$ and $D[u,v] > \phi$ then the constraint $W[u,w] + r[w] - r[u] \geq 1$ is redundant, since there are already $W[u,v] + r[v] - r[u] \geq 1$ and $w[v,w] + r[w] - r[v] \geq 0$. However, it may not be easy to check and remove all redundancy in the constraints.

In order to build the minimum cost flow network, it is needed to first compute both matrices W and D. Since $W[u,v]$ is the shortest path from u to v in terms of w, the computation of W can be done by an all-pair shortest paths algorithm such as Floyd–Warshall's algorithm [1]. Furthermore, if the ordered pair $(w[x,y], -d[x])$ is used as the edge weight for each $(x,y) \in E$, an all-pair shortest paths algorithm can also be used to compute both W and D. The algorithm will add weights by component-wise addition and will compare weights by lexicographic ordering.

Leiserson and Saxe [3]'s first algorithm for the minimum period retiming was also based on the matrices W and D. The idea was that the constraints in the integer linear program for the minimum area retiming can be checked efficiently by Bellman–Ford's shortest paths algorithm [1], since they are just difference inequalities. This gives a feasibility checking for any given clock period φ. Then the optimal clock period can be found by a binary search on a range of possible periods. The feasibility checking can be done in $O(|V|^3)$ time, thus the runtime of such an algorithm is $O(|V|^3 \log |V|)$.

Their second algorithm got rid of the construction of the matrices W and D. It still used a clock period feasibility checking within a binary search. However, the feasibility checking was done by incremental retiming. It works as follows. Starting with $r = 0$, the algorithm computes the arrival time of each node by the longest paths computation on a DAG (Directed Acyclic Graph). For each node v with an arrival time larger than the given period φ, the $r[v]$ will be increased by one. The process of the arrival time computation and r increasing will be repeated $|V| - 1$ times. After that, if there is still arrival time that is larger than φ, then the period is infeasible. Since the feasibility checking is done in $O(|V||E|)$ time, the runtime for the minimum period retiming is $O(|V||E| \log |V|)$.

Applications

Shenoy and Rudell [7] implemented Leiserson and Saxe's minimum period and minimum area retiming algorithms with some efficiency improvements. For minimum period retiming, they implemented the second algorithm and, in order to find out infeasibility earlier, they introduced a pointer from one node to another where at least one register is required between them. A cycle formed by the pointers indicates the infeasibility of the given period. For minimum area retiming, they removed some of the redundancy in the constraints and used the cost-scaling algorithm of Goldberg and Tarjan [2] for the minimum cost flow computation.

Open Problems

As can be seen from the second minimum period retiming algorithm here and Zhou's algorithm [8] in another entry (► Circuit Retiming: An Incremental Approach), incremental computation of the longest combinational paths (i. e. those without register on them) is more efficient than constructing the dense graph (via matrices W and D). However, the minimum area retiming algorithm is still based on a minimum cost network flow on the dense graph. An interesting open question is to see whether a more efficient algorithm based on incremental retiming can be designed for the minimum area problem.

Experimental Results

Sapatnekar and Deokar [6] and Pan [5] proposed continuous retiming as an efficient approximation for minimum period retiming, and reported the experimental results. Maheshwari and Sapatnekar [4] also proposed some efficiency improvements to the minimum area retiming algorithm and reported their experimental results.

Cross References

► Circuit Retiming: An Incremental Approach

Recommended Reading

1. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
2. Goldberg, A.V., Tarjan, R.E.: Solving minimum cost flow problem by successive approximation. In: Proc. ACM Symposium on the Theory of Computing, pp. 7–18 (1987). Full paper in: Math. Oper. Res. **15**, 430–466 (1990)
3. Leiserson, C.E., Saxe, J.B.: Retiming synchronous circuitry. Algorithmica **6**, 5–35 (1991)
4. Maheshwari, N., Sapatnekar, S.S.: Efficient retiming of large circuits, IEEE Transactions on Very Large-Scale Integrated Systems. **6**, 74–83 (1998)
5. Pan, P.: Continuous retiming: Algorithms and applications. In: Proc. Intl. Conf. Comput. Design, pp. 116–121. IEEE Press, Los Almitos (1997)
6. Sapatnekar, S.S., Deokar, R.B.: Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits. IEEE Trans. Comput. Aided Des. **15**, 1237–1248 (1996)
7. Shenoy, N., Rudell, R.: Efficient implementation of retiming. In Proc. Intl. Conf. Computer-Aided Design, pp. 226–233. IEEE Press, Los Almitos (1994)

8. Zhou, H.: Deriving a new efficient algorithm for min-period retiming. In Asia and South Pacific Design Automation Conference, Shanghai, China, Jan. ACM Press, New York (2005)

Circuit Retiming: An Incremental Approach

2005; Zhou

HAI ZHOU
Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA

Keywords and Synonyms

Minimum period retiming; Min-period retiming

Problem Definition

Circuit retiming is one of the most effective structural optimization techniques for sequential circuits. It moves the registers within a circuit without changing its function. The minimal period retiming problem needs to minimize the longest delay between any two consecutive registers, which decides the clock period.

The problem can be formally described as follows. Given a directed graph $G = (V, E)$ representing a circuit – each node $v \in V$ represents a gate and each edge $e \in E$ represents a signal passing from one gate to another – with gate delays $d\colon V \to \mathbb{R}^+$ and register numbers $w\colon E \to \mathbb{N}$, it asks for a relocation of registers $w'\colon E \to \mathbb{N}$ such that the maximal delay between two consecutive registers is minimized.

Notations To guarantee that the new registers are actually a relocation of the old ones, a label $r\colon V \to \mathbb{Z}$ is used to represent how many registers are moved from the outgoing edges to the incoming edges of each node. Using this notation, the new number of registers on an edge (u,v) can be computed as

$$w'[u, v] = w[u, v] + r[v] - r[u] .$$

Furthermore, to avoid explicitly enumerating the paths in finding the longest path, another label $t\colon V \to \mathbb{R}^+$ is introduced to represent the output arrival time of each gate, that is, the maximal delay of a gate from any preceding register. The condition for t to be at least the combinational delays is

$$\forall (u, v) \in E\colon w'[u, v] = 0 \Rightarrow t[v] \geq t[u] + d[v] .$$

Constraints and Objective Based on the notations, a valid retiming r should not have any negative number of registers on any edge. Such a validity condition is given as

$$P0(r) \triangleq \forall (u, v) \in E\colon w[u, v] + r[v] - r[u] \geq 0 .$$

As already stated, the conditions for t to be valid arrival time is given by the following two predicates.

$$P1(t) \triangleq \forall v \in V\colon t[v] \geq d[v]$$

$$P2(r, t) \triangleq \forall (u, v) \in E\colon r[u] - r[v] = w[u, v] \Rightarrow t[v] - t[u] \geq d[v] .$$

A predicate P is used to denote the conjunction of the above conditions:

$$P(r, t) \triangleq P0(r) \wedge P1(t) \wedge P2(r, t) .$$

A minimal period retiming is a solution $\langle r, t\rangle$ satisfying the following optimality condition:

$$P3 \triangleq \forall r', t'\colon P(r', t') \Rightarrow \max(t) \leq \max(t') ,$$

where

$$\max(t) \triangleq \max_{v \in V} t[v] .$$

Since only a valid retiming (r', t') will be discussed in the sequel, to simplify the presentation, the range condition $P(r', t')$ will often be omitted; the meaning shall be clear from the context.

Key Results

This section will show how an efficient algorithm is designed for the minimal period retiming problem. Contrary to the usual way of only presenting the final product, i. e. the algorithm, but not the ideas on its design, a step-by-step design process will be shown to finally arrive at the algorithm.

To design an algorithm is to construct a procedure such that it will terminate in finite steps and will satisfy a given predicate when it terminates. In the minimal period retiming problem, the predicate to be satisfied is $P0 \wedge P1 \wedge P2 \wedge P3$. The predicate is also called the *post-condition*. It can be argued that any non-trivial algorithm will have at least one loop, otherwise, the processing length is only proportional to the text length. Therefore, some part of the post-condition will be iteratively satisfied by the loop, while the remaining part will be initially satisfied by an initialization and made invariant during the loop.

The first decision needed to make is to partition the post-condition into possible invariant and loop goal. Among the four conjuncts, the predicate $P3$ gives the optimality condition and is the most complex one. Therefore,

it will be used as a loop goal. On the other hand, the predicates $P0$ and $P1$ can be easily satisfied by the following simple initialization.

$$r, t := 0, d \ .$$

Based on these, the plan is to design an algorithm with the following scheme.

$$\begin{aligned} &r, t := 0, d \\ &\text{do}\{P0 \wedge P1\} \\ &\quad \neg P2 \rightarrow \text{update } t \\ &\quad \neg P3 \rightarrow \text{update } r \\ &\text{od}\{P0 \wedge P1 \wedge P2 \wedge P3\} \ . \end{aligned}$$

The first command in the loop can be refined as

$$\exists (u, v) \in E \colon r[u] - r[v] = w[u, v] \wedge t[v] - t[u] < d[v] \rightarrow t[v] := t[u] + d[v] \ .$$

This is simply the Bellman–Ford relaxations for computing the longest paths.

The second command is more difficult to refine. If $\neg P3$, that is, there exists another valid retiming $\langle r', t' \rangle$ such that $\max(t) > \max(t')$, then on any node v such that $t[v] = \max(t)$ it must have $t'[v] < t[v]$. One property known on these nodes is

$$\forall v \in V \colon t'[v] < t[v] \Rightarrow (\exists u \in V \colon r[u] - r[v] > r'[u] - r'[v]) \ ,$$

which means that if the arrival time of v is smaller in another retiming $\langle r', t' \rangle$, then there must be a node u such that r' gives more registers between u and v. In fact, one such a u is the starting node of the longest combinational path to v that gives the delay of $t[v]$.

To reduce the clock period, the variable r needs to be updated to make it closer to r'. It should be noted that it is not the absolute values of r but their differences that are relevant in the retiming. If $\langle r, t \rangle$ is a solution to a retiming problem, then $\langle r + c, t \rangle$, where $c \in \mathbb{Z}$ is an arbitrary constant, is also a solution. Therefore r can be made "closer" to r' by allocating more registers between u and v, that is, by either decreasing $r[u]$ or increasing $r[v]$. Notice that v can be easily identified by $t[v] = \max(t)$. No matter whether $r[v]$ or $r[u]$ is selected to change, the amount of change should be only one since r should not be over-adjusted. Thus, after the adjustment, it is still true that $r[v] - r[u] \leq r'[v] - r'[u]$, or equivalently $r[v] - r'[v] \leq r[u] - r'[u]$. Since v is easy to identify, $r[v]$ is selected to increase. The arrival time $t[v]$ can be immediately reduced to $d[v]$. This gives a refinement of the second commend:

$$\neg P3 \wedge P2 \wedge \exists v \in V \colon t[v] = \max(t) \rightarrow r[v], t[v] := r[v] + 1, d[v] \ .$$

Since registers are moved in the above operation, the predicate $P2$ may be violated. However, the first command will take care of it. That command will increase t on some nodes; some may even become larger than $\max(t)$ before the register move. The same reasoning using $\langle r', t' \rangle$ shows that their r values shall be increased, too. Therefore, to implement this As-Soon-As-Possible (ASAP) increase of r, a snapshot of $\max(t)$ needs to be taken when $P2$ is valid. Physically, such a snapshot records one feasible clock period ϕ, and can be implemented by adding one more command in the loop:

$$P2 \wedge \phi > \max(t) \rightarrow \phi := \max(t) \ .$$

However, such an ASAP operation may increase $r[u]$ even when $w[u, v] - r[u] + r[v] = 0$ for an edge (u,v). It means that $P0$ may no longer be an invariant. But moving $P0$ from invariant to loop goal will not cause a problem since one more command can be added in the loop to take care of it:

$$\exists (u, v) \in E \colon r[u] - r[v] > w[u, v] \rightarrow r[v] := r[u] - w[u, v] \ .$$

Putting all things together, the algorithm now has the following form.

$$\begin{aligned} &r, t, \phi := 0, d, \infty; \\ &\text{do}\{P1\} \\ &\quad \exists (u, v) \in E \colon r[u] - r[v] = w[u, v] \\ &\qquad \wedge\, t[v] - t[u] < d[v] \rightarrow t[v] := t[u] + d[v] \\ &\quad \neg P3 \wedge \exists v \in V \colon t[v] \geq \phi \\ &\qquad \rightarrow r[v], t[v] := r[v] + 1, d[v] \\ &\quad P0 \wedge P2 \wedge \phi > \max(t) \rightarrow \phi := \max(t) \\ &\quad \exists (u, v) \in E \colon r[u] - r[v] > w[u, v] \\ &\qquad \rightarrow r[v] := r[u] - w[u, v] \\ &\text{od}\{P0 \wedge P1 \wedge P2 \wedge P3\} \ . \end{aligned}$$

The remaining task to complete the algorithm is how to check $\neg P3$. From previous discussion, it is already known that $\neg P3$ implies that there is a node u such that $r[u] - r'[u] \geq r[v] - r'[v]$ every time after $r[v]$ is increased. This means that $\max_{v \in V} r[v] - r'[v]$ will not increase. In

Circuit Retiming: An Incremental Approach, Table 1
Experimental Results

name	#gates	clock period		$\sum r$	#updates	time(s)	ASTRA	
		before	after				A(s)	B(s)
s1423	490	166	127	808	7619	0.02	0.03	0.02
s1494	558	89	88	628	7765	0.02	0.01	0.01
s9234	2027	89	81	2215	76943	0.12	0.11	0.09
s9234.1	2027	89	81	2164	77644	0.16	0.11	0.10
s13207	2573	143	82	4086	28395	0.12	0.38	0.12
s15850	3448	186	77	12038	99314	0.36	0.43	0.17
s35932	12204	109	100	16373	108459	0.28	0.24	0.65
s38417	8709	110	56	9834	155489	0.58	0.89	0.64
s38584	11448	191	163	19692	155637	0.41	0.50	0.67
s38584.1	11448	191	183	9416	114940	0.48	0.55	0.78

other words, there is at least one node v whose $r[v]$ will not change. Before $r[v]$ is increased, it also has $w_{u \rightsquigarrow v} - r[u] + r[v] \leq 0$, where $w_{u \rightsquigarrow v} \geq 0$ is the original number of registers on one path from u to v, which gives $r[v] - r[u] \leq 1$ even after the increase of $r[v]$. This implies that there will be at least $i + 1$ nodes whose r is at most i for $0 \leq i < |V|$. In other words, the algorithm can keep increasing r and when there is any r reaching $|V|$ it shows that $P3$ is satisfied. Therefore, the complete algorithm will have the following form.

$r, t, \phi := 0, d, \infty;$
do$\{P1\}$
$\quad \exists (u, v) \in E\colon r[u] - r[v] = w[u, v]$
$\qquad \wedge\, t[v] - t[u] < d[v] \rightarrow t[v] := t[u] + d[v]$
$\quad (\forall v \in V\colon r[v] < |V|)$
$\qquad \wedge\, \exists v \in V\colon t[v] \geq \phi \rightarrow r[v], t[v] := r[v] + 1, d[v]$
$\quad (\exists v \in V\colon r[v] \geq |V|)$
$\qquad \wedge\, \exists v \in V\colon t[v] > \phi \rightarrow r[v], t[v] := r[v] + 1, d[v]$
$\quad P0 \wedge P2 \wedge \phi > \max(t) \rightarrow \phi := \max(t)$
$\quad \exists (u, v) \in E\colon r[u] - r[v] > w[u, v]$
$\qquad \rightarrow r[v] := r[u] - w[u, v]$
od$\{P0 \wedge P1 \wedge P2 \wedge P3\}$.

The correctness of the algorithm can be proved easily by showing that the invariant $P1$ is maintained and the negation of the guards implies $P0 \wedge P2 \wedge P3$. The termination is guaranteed by the monotonic increase of r and an upper bound on it. In fact, the following theorem gives its worst case runtime.

Theorem 1 *The worst case running time of the given retiming algorithm is upper bounded by* $O(|V|^2|E|)$.

The runtime bound of the retiming algorithm is got under the worst case assumption that each increase on r will trigger a timing propagation on the whole circuit ($|E|$ edges). This is only true when the r increase moves all registers in the circuit. However, in such a case, the r is upper bounded by 1, thus the running time is not larger than $O(|V||E|)$. On the other hand, when the r value is large, the circuit is partitioned by the registers into many small parts, thus the timing propagation triggered by one r increase is limited within a small tree.

Applications

In the basic algorithm, the optimality $P3$ is verified by an $r[v] \geq |V|$. However, in most cases, the optimality condition can be discovered much earlier. Since each time $r[v]$ is increased, there must be a "safe-guard" node u such that $r[u] - r'[u] \geq r[v] - r'[v]$ after the operation. Therefore, if a pointer is introduced from v to u when $r[v]$ is increased, the pointers cannot form a cycle under $\neg P3$. In fact, the pointers will form a forest where the roots have $r = 0$ and a child can have an r at most one larger than its parent. Using a cycle by the pointers as an indication of $P3$, instead of an $r[v] \geq |V|$, the algorithm can have much better practical performance.

Open Problems

Retiming is usually used to optimize either the clock period or the number of registers in the circuit. The discussed algorithm solves only the minimal period retiming problem. The retiming problem for minimizing the number of registers under a given period has been solved by Leiserson and Saxe [1] and is presented in another entry in this encyclopedia. Their algorithm reduces the problem to the dual of a minimal cost network problem on a denser graph. An interesting open question is to see whether an efficient iterative algorithm similar to Zhou's algorithm can be designed for the minimal register problem.

Experimental Results

Experimental results are reported by Zhou [3] which compared the runtime of the algorithm with an efficient heuristic called ASTRA [2]. The results on the ISCAS89 benchmarks are reproduced here in Table 1 from [3], where columns *A* and *B* are the running time of the two stages in ASTRA.

Cross References

► Circuit Retiming

Recommended Reading

1. Leiserson, C.E., Saxe, J.B.: Retiming synchronous circuitry. Algorithmica **6**, 5–35 (1991)
2. Sapatnekar, S.S., Deokar, R.B.: Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits. IEEE Transactions on Computer Aided Design **15**, 1237–1248 (1996)
3. Zhou, H.: Deriving a new efficient algorithm for min-period retiming. In: Asia and South Pacific Design Automation Conference, Shanghai, China, January 2005

Clock Synchronization

1994; Patt-Shamir, Rajsbaum

Boaz Patt-Shamir
Department of Electrical Engineering,
Tel-Aviv University, Tel-Aviv, Israel

Problem Definition

Background and Overview

Coordinating processors located in different places is one of the fundamental problems in distributed computing. In his seminal work, Lamport [4,5] studied the model where the only source of coordination is message exchange between the processors; the time that elapses between successive steps at the same processor, as well as the time spent by a message in transit, may be arbitrarily large or small. Lamport observed that in this model, called the *asynchronous model*, temporal concepts such as "past" and "future" are derivatives of causal dependence, a notion with a simple algorithmic interpretation. The work of Patt-Shamir and Rajsbaum [10] can be viewed as extending Lamport's qualitative treatment with quantitative concepts. For example, a statement like "event *a* happened before event *b*" may be refined to a statement like "event *a* happened at least 2 time units and at most 5 time units before event *b*". This is in contrast to most previous theoretical work, which focused on the linear-programming aspects of clock synchronization (see below).

The basic idea in [10] is as follows. First, the framework is extended to allow for upper and lower bounds on the time that elapses between pairs of events, using the system's *real-time specification*. The notion of real-time specification is a very natural one. For example, most processors have local clocks, whose rate of progress is typically bounded with respect to real time (these bounds are usually referred to as the clock's "drift bounds"). Another example is send and receive events of a given message: It is always true that the receive event occurs before the send event, and in many cases, tighter lower and upper bounds are available. Having defined real-time specification, [10] proceeds to show how to combine these local bounds global bounds in the best possible way using simple graph-theoretic concepts. This allows one to derive optimal protocols that say, for example, what is the current reading of a remote clock. If that remote clock is the standard clock, then the result is optimal clock synchronization in the common sense (this concept is called "external synchronization" below).

Formal Model

The system consists of a fixed set of interconnected *processors*. Each processor has a *local clock*. An *execution* of the system is a sequence of events, where each event is either a *send* event, a *receive* event, or an *internal* event. Regarding communication, it is only assumed that each receive event of a message m has a unique corresponding send event of m. This means that messages may be arbitrarily lost, duplicated or reordered, but not corrupted. Each event e occurs at a single specified processor, and has two real numbers associated with it: its *local time*, denoted $\mathrm{LT}(e)$, and its *real time*, denoted $\mathrm{RT}(e)$. The local time of an event models the reading of the local clock when that event occurs, and the local processor may use this value, e. g., for calculations, or by sending it over to another processor. By contrast, the real time of an event is not observable by processors: it is an abstract concept that exists only in the analysis.

Finally, the real-time properties of the system are modeled by a pair of functions that map each pair of events to $\mathbb{R} \cup \{-\infty, \infty\}$: given two events e and e', $L(e, e') = \ell$ means that $\mathrm{RT}(e') - \mathrm{RT}(e) \geq \ell$, and $H(e, e') = h$ means that $\mathrm{RT}(e') - \mathrm{RT}(e) \leq h$, i. e., that the number of (real) time units since the occurrence of event e until the occurrence of e' is at least ℓ and at most h. Without loss of generality, it is assumed that $L(e, e') = -H(e', e)$ for all events e, e' (just use the smaller of them). Henceforth, only the

upper bounds function H is used to represent the real-time specification.

Some special cases of real time properties are particularly important. In a completely asynchronous system, $H(e', e) = 0$ if either e occurs before e' in the same processor, or if e and e' are the send and receive events, respectively, of the same message. (For simplicity, it is assumed that two ordered events may have the same real time of occurrence.) In all other cases $H(e, e') = \infty$. On the other extreme of the model spectrum, there is the *drift-free* clocks model, where all local clocks run at exactly the rate of real time. Formally, in this case $H(e, e') = \mathrm{LT}(e') - \mathrm{LT}(e)$ for any two events e and e' occurring at the same processor. Obviously, it may be the case that only some of the clocks in the system are drift-free.

Algorithms

In this work, message generation and delivery is completely decoupled from message information. Formally, messages are assumed to be generated by some "send module", and delivered by the "communication system". The task of algorithms is to add contents in messages and state variables in each node. (The idea of decoupling synchronization information from message generation was introduced in [1].) The algorithm only has local information, i. e., contents of the local state variables and the local clock, as well as the contents of the incoming message, if we are dealing with a receive event. It is also assumed that the real time specification is known to the algorithm. The conjunction of the events, their and their local times (but not their real times) is called as the *view* of the given execution. Algorithms, therefore, can only use as input the view of an execution and its real time specification.

Problem Statement

The simplest variant of clock synchronization is *external synchronization*, where one of the processors, called the source, has a drift-free clock, and the task of all processors is to maintain the tightest possible estimate on the current reading of the source clock. This formulation corresponds to the Newtonian model, where the processors reside in a well-defined time coordinate system, and the source clock is reading the standard time. Formally, in external synchronization each processor v has two output variables Δ_v and ε_v; the estimate of v of the source time at a given state is $LT_v + \Delta_v$, where LT_v is the current local time at v. The algorithm is required to guarantee that the difference between the source time and it estimate is at most ε_v (note that Δ_v, as well as ε_v, may change dynamically during the execution). The performance of the algorithm is judged by the value of the ε_v variables: the smaller, the better.

In another variant of the problem, called *internal synchronization*, there is no distinguished processor, and the requirement is essentially that all clocks will have values which are close to each other. Defining this variant is not as straightforward, because trivial solutions (e. g., "set all clocks to 0 all the time") must be disqualified.

Key Results

The key construct used in [10] is the *synchronization graph* of an execution, defined by combining the concepts of local times and real-time specification as follows.

Definition 1 Let β be a view of an execution of the system, and let H be a real time specification for β. The *synchronization graph* generated by β and H is a directed weighted graph $\Gamma_{\beta H} = (V, E, w)$, where V is the set of events in β, and for each ordered pair of events $p\,q$ in β such that $H(p, q) < \infty$, there is a directed edge $(p, q) \in E$. The *weight* of an edge (p, q) is $w(p, q) \stackrel{\text{def}}{=} H(p, q) - \mathrm{LT}(p) + \mathrm{LT}(q)$.

The natural concept of *distance* from an event p to an event q in a synchronization graph Γ, denoted $d_\Gamma(p, q)$, is defined by the length of the shortest weight path from p to q, or infinity if q is not reachable from p. Since weights may be negative, one has to prove that the concept is well defined: indeed, it is shown that if $\Gamma_{\beta H}$ is derived from an execution with view β that satisfies real time specification H, then $\Gamma_{\beta H}$ does not contain directed cycles of negative weight.

The main algorithmic result concerning synchronization graphs is summarized in the following theorem.

Theorem 1 *Let α be an execution with view β. Then α satisfies the real time specification H if and only if* $\mathrm{RT}(p) - \mathrm{RT}(q) \le d_\Gamma(p, q) + \mathrm{LT}(p) - \mathrm{LT}(q)$ *for any two events p and q in $\Gamma_{\beta H}$.*

Note that all quantities in the r.h.s. of the inequality are available to the synchronization algorithm, which can therefore determine upper bounds on the real time that elapses between events. Moreover, these bounds are the best possible, as implied by the next theorem.

Theorem 2 *Let $\Gamma_{\beta H} = (V, E, w)$ be a synchronization graph obtained from a view β satisfying real time specification H. Then for any given event $p_0 \in V$, and for any finite number $N > 0$, there exist executions α_0 and α_1 with view β, both satisfying H, and such that the following real time assignments hold.*

- *In* α_0, *for all* $q \in V$ *with* $d_\Gamma(q, p_0) < \infty$, $\mathrm{RT}_{\alpha_0}(q) = \mathrm{LT}(q) + d_\Gamma(q, p_0)$, *and for all* $q \in V$ *with* $d_\Gamma(q, p_0) = \infty$, $\mathrm{RT}_{\alpha_0}(q) > \mathrm{LT}(q) + N$.
- *In* α_1, *for all* $q \in V$ *with* $d_\Gamma(p_0, q) < \infty$, $\mathrm{RT}_{\alpha_1}(q) = \mathrm{LT}(q) - d_\Gamma(p_0, q)$, *and for all* $q \in V$ *with* $d_\Gamma(p_0, q) = \infty$, $\mathrm{RT}_{\alpha_1}(q) < \mathrm{LT}(q) - N$.

From the algorithmic viewpoint, one important drawback of results of Theorems 1 and 2 is that they depend on the view of an execution, which may grow without bound. Is it really necessary? The last general result in [10] answers this question in the affirmative. Specifically, it is shown that in some variant of the *branching program* computational model, the space complexity of any synchronization algorithm that works with arbitrary real time specifications cannot be bounded by a function of the system size. The result is proved by considering multiple scenarios on a simple system of four processors on a line.

Later Developments

Based on the concept of synchronization graph, Ostrovsky and Patt-Shamir present a refined general optimal algorithm for clock synchronization [9]. The idea in [9] is to discard parts of the synchronization graphs that are no longer relevant. Roughly speaking, the complexity of the algorithm is bounded by a polynomial in the system size and the ratio of processors speeds.

Much theoretical work was invested in the internal synchronization variant of the problem. For example, Lundelius and Lynch [7] proved that in a system of n processors with full connectivity, if message delays can take arbitrary values in $[0, 1]$ and local clocks are drift-free, then the best synchronization that can be guaranteed is $1 - \frac{1}{n}$. Helpern et al. [3] extended their result to general graphs using linear-programming techniques. This work, in turn, was extended by Attiya et al. [1] to analyze any given execution (rather than only the worst case for a given topology), but the analysis is performed off-line and in a centralized fashion. The work of Patt-Shamir and Rajsbaum [11] extended the "per execution" viewpoint to online distributed algorithms, and shifted the focus of the problem to external synchronization.

Recently, Fan and Lynch [2] proved that in a line of n processors whose clocks may drift, no algorithm can guarantee that the difference between the clock readings of all pairs of neighbors is $o(\log n / \log \log n)$.

Clock synchronization is very useful in practice. See, for example, Liskov [6] for some motivation. It is worth noting that the Internet provides a protocol for external clock synchronization called NTP [8].

Applications

Theorem 1 immediately gives rise to an algorithm for clock synchronization: every processor maintains a representation of the synchronization graph portion known to it. This can be done using a full information protocol: In each outgoing message this graph is sent, and whenever a message arrives, the graph is extended to include the new information from the graph in the arriving message. By Theorem 2, the synchronization graph obtained this way represents at any point in time all information available required for optimal synchronization. For example, consider external synchronization. Directly from definitions it follows that all events associated with a drift-free clock (such as events in the source node) are at distance 0 from each other in the synchronization graph, and can therefore be considered, for distance computations, as a single node s. Now, assuming that the source clock actually shows real time, it is easy to see that for any event p,

$$\mathrm{RT}(p) \in [\mathrm{LT}(p) - d(s, p), \mathrm{LT}(p) + d(p, s)] ,$$

and furthermore, no better bounds can be obtained by any correct algorithm.

The general algorithm described above (maintaining the complete synchronization graph) can be used also to obtain optimal results for internal synchronization; details are omitted.

An interesting special case is where all clocks are drift free. In this case, the size of the synchronization graph remains fixed: similarly to a source node in external synchronization, all events occurring at the same processor can be mapped to a single node; parallel edges can be replaced by a single new edge whose weight is minimal among all old edges. This way one can obtain a particularly efficient distributed algorithm solving external clock synchronization, based on the distributed Bellman–Ford algorithm for distance computation.

Finally, note that the asynchronous model may also be viewed as a special case of this general theory, where an event p "happens before" an event q if and only if $d(p, q) \leq 0$.

Open Problems

One central issue in clock synchronization is faulty executions, where the real time specification is violated. Synchronization graphs detect any detectable error: views which do not have an execution that conforms with the real time specification will result in synchronization graphs with negative cycles. However, it is desirable to overcome such faults, say by removing from the synchro-

nization graph some edges so as to break all negative-weight cycles. The natural objective in this case is to remove the least number of edges. This problem is APX-hard as it generalizes the Feedback Arc Set problem. Unfortunately, no non-trivial approximation algorithms for it are known.

Cross References

► Causal Order, Logical Clocks, State Machine Replication

Recommended Reading

1. Attiya, H., Herzberg, A., Rajsbaum, S.: Optimal clock synchronization under different delay assumptions. SIAM J. Comput. **25**(2), 369–389 (1996)
2. Fan, R., Lynch, N.A.: Gradient clock synchronization. Distrib. Comput. **18**(4), 255–266 (2006)
3. Halpern, J.Y., Megiddo, N., Munshi, A.A.: Optimal precision in the presence of uncertainty. J. Complex. **1**, 170–196 (1985)
4. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7), 558–565 (1978)
5. Lamport, L.: The mutual exclusion problem. Part I: A theory of interprocess communication. J. ACM **33**(2), 313–326 (1986)
6. Liskov, B.: Practical uses of synchronized clocks in distributed systems. Distrib. Comput. **6**, 211–219 (1993). Invited talk at the 9th Annual ACM Symposium on Principles of Distributed Computing, Quebec City 22–24 August 1990
7. Lundelius, J., Lynch, N.: A new fault-tolerant algorithm for clock synchronization. Inf. Comput. **77**, 1–36 (1988)
8. Mills, D.L.: Computer Network Time Synchronization: The Network Time Protocol. CRC Press, Boca Raton (2006)
9. Ostrovsky, R., Patt-Shamir, B.: Optimal and efficient clock synchronization under drifting clocks. In: Proceedings of the 18th Annual Symposium on Principles of Distributed Computing, pp. 3–12, Atlanta, May (1999)
10. Patt-Shamir, B., Rajsbaum, S.: A theory of clock synchronization. In: Proceedings of the 26th Annual ACM Symposium on Theory of Computing, pp. 810–819, Montreal, May (1994)
11. Patt-Shamir, B., Rajsbaum, S.: A theory of clock synchronization. In: Proceedings of the 26th Annual ACM Symposium on Theory of Computing, pp. 810–819, Montreal 23–25 May 1994

Closest String and Substring Problems

2002; Li, Ma, Wang

Lusheng Wang
Department of Computer Science,
City University of Hong Kong, Hong Kong, China

Problem Definition

The problem of finding a center string that is "close" to every given string arises and has applications in computational molecular biology and coding theory.

This problem has two versions: The first problem comes from coding theory when we are looking for a code not too far away from a given set of codes.

Problem 1 (The closest string problem)
INPUT: *a set of strings* $S = \{s_1, s_2, \ldots, s_n\}$, *each of length* m.
OUTPUT: *the smallest* d *and a string* s *of length* m *which is within Hamming distance* d *to each* $s_i \in S$.

The second problem is much more elusive than the Closest String problem. The problem is formulated from applications in finding conserved regions, genetic drug target identification, and genetic probes in molecular biology.

Problem 2 (The closest substring problem)
INPUT: *an integer* L *and a set of strings* $S = \{s_1, s_2, \ldots, s_n\}$, *each of length* m.
OUTPUT: *the smallest* d *and a string* s, *of length* L, *which is within Hamming distance* d *away from a length* L *substring* t_i *of* s_i *for* $i = 1, 2, \ldots n$.

Key Results

The following results are from [1].

Theorem 1 *There is a polynomial time approximation scheme for the closest string problem.*

Theorem 2 *There is a polynomial time approximation scheme for the closest substring problem.*

Results for other measures can be found in [10,11,12].

Applications

Many problems in molecular biology involve finding similar regions common to each sequence in a given set of DNA, RNA, or protein sequences. These problems find applications in locating binding sites and finding conserved regions in unaligned sequences [2,7,9,13,14], genetic drug target identification [8], designing genetic probes [8], universal PCR primer design [4,8], and, outside computational biology, in coding theory [5,6]. Such problems may be considered to be various generalizations of the common substring problem, allowing errors. Many measures have been proposed for finding such regions common to every given string. A popular and one of the most fundamental measures is the Hamming distance. Moreover, two popular objective functions are used in these areas. One is the total sum of distances between the center string (common substring) and each of the given strings. The other is the maximum distance between the center string and a given string. For more details, see [8].

A more General Problem

The *distinguishing substring selection* problem has as input two sets of strings, B and G. It is required to find a substring of unspecified length (denoted by L) such that it is, informally, close to a substring of every string in B and far away from every length L substring of strings in $\mathcal{G}$. However, we can go through all the possible L and we may assume that every string in $\mathcal{G}$ has the same length L since $\mathcal{G}$ can be reconstructed to contain all substrings of length L in each of the good strings.

The problem is formally defined as follows: Given a set $\mathcal{B} = \{s_1, s_2, \ldots, s_{n_1}\}$ of n_1 (bad) strings of length at least L, and a set $\mathcal{G} = \{g_1, g_2, \ldots g_{n_2}\}$ of n_2 (good) strings of length exactly L, as well as two integers d_b and d_g ($d_b \leq d_g$), the distinguishing substring selection problem ($DSSP$) is to find a string s such that for each string $s_i \in \mathcal{B}$ there exists a length-L substring t_i of s_i with $d(s, t_i) \leq d_b$ and for any string $g_i \in \mathcal{G}$, $d(s, g_i) \geq d_g$. Here $d(,)$ represents the Hamming distance between two strings. If all strings in $\mathcal{B}$ are also of the same length L, the problem is called the distinguishing string problem (DSP).

The distinguishing string problem was first proposed in [8] for generic drug target design. The following results are from [3].

Theorem 3 *There is a polynomial time approximation scheme for the distinguishing substring selection problem. That is, for any constant $\epsilon > 0$, the algorithm finds a string s of length L such that for every $s_i \in \mathcal{B}$, there is a length-L substring t_i of s_i with $d(t_i, s) \leq (1+\epsilon)d_b$ and for every substring u_i of length L of every $g_i \in \mathcal{G}$, $d(u_i, s) \geq (1-\epsilon)d_g$, if a solution to the original pair $(d_b \leq d_g)$ exists. Since there are a polynomial number of such pairs (d_b, d_g), we can exhaust all the possibilities in polynomial time to find a good approximation required by the corresponding application problems.*

Open Problems

The PTAS's designed here use linear programming and randomized rounding technique to solve some cases for the problem. Thus, the running time complexity of the algorithms for both the closest string and closest substring is very high. An interesting open problem is to design more efficient PTAS's for both problems.

Cross References

- ▶ Closest Substring
- ▶ Efficient Methods for Multiple Sequence Alignment with Guaranteed Error Bounds
- ▶ Engineering Algorithms for Computational Biology
- ▶ Multiplex PCR for Gap Closing (Whole-genome Assembly)

Recommended Reading

1. Ben-Dor, A., Lancia, G., Perone, J., Ravi, R.: Banishing bias from consensus sequences. In: Proc. 8th Ann. Combinatorial Pattern Matching Conf., pp. 247–261. (1997)
2. Deng, X., Li, G., Li, Z., Ma, B., Wang, L.: Genetic Design of Drugs Without Side-Effects. SIAM. J. Comput. **32**(4), 1073–1090 (2003)
3. Dopazo, J., Rodríguez, A., Sáiz, J.C., Sobrino, F.: Design of primers for PCR amplification of highly variable genomes. CABIOS **9**, 123–125 (1993)
4. Frances, M., Litman, A.: On covering problems of codes. Theor. Comput. Syst. **30**, 113–119 (1997)
5. Gąsieniec, L., Jansson, J., Lingas, A.: Efficient approximation algorithms for the hamming center problem. In: Proc. 10th ACM-SIAM Symp. on Discrete Algorithms., pp. 135–S906. (1999)
6. Hertz, G., Stormo, G.: Identification of consensus patterns in unaligned DNA and protein sequences: a large-deviation statistical basis for penalizing gaps. In: Proc. 3rd Int'l Conf. Bioinformatics and Genome Research, pp. 201–216. (1995)
7. Lanctot, K., Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing string selection problems. In: Proc. 10th ACM-SIAM Symp. on Discrete Algorithms, pp. 633–642. (1999)
8. Lawrence, C., Reilly, A.: An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. Proteins **7**, 41–51 (1990)
9. Li, M., Ma, B., Wang, L.: On the closest string and substring problems. J. ACM **49**(2), 157–171 (2002)
10. Li, M., Ma, B., Wang, L.: Finding similar regions in many sequences. J. Comput. Syst. Sci. (1999)
11. Li, M., Ma, B., Wang, L.: Finding similar regions in many strings. In: Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing, pp. 473–482. Atlanta (1999)
12. Ma, B.: A polynomial time approximation scheme for the closest substring problem. In: Proc. 11th Annual Symposium on Combinatorial Pattern Matching, Montreal, pp. 99–107. (2000)
13. Stormo, G.: Consensus patterns in DNA. In: Doolittle, R.F. (ed.) Molecular evolution: computer analysis of protein and nucleic acid sequences. Methods in Enzymology, vol. 183, pp. 211–221 (1990)
14. Stormo, G., Hartzell III, G.W.: Identifying protein-binding sites from unaligned DNA fragments. Proc. Natl. Acad. Sci. USA. **88**, 5699–5703 (1991)

Closest Substring

2005; Marx

Jens Gramm
WSI Institute of Theoretical Computer Science, Tübingen University, Tübingen, Germany

Keywords and Synonyms

Common approximate substring

Problem Definition

CLOSEST SUBSTRING is a core problem in the field of consensus string analysis with, in particular, applications in computational biology. Its decision version is defined as follows.

CLOSEST SUBSTRING
Input: k strings $s_1, s_2, \ldots, s_k$ over alphabet Σ and non-negative integers d and L.
Question: Is there a string s of length L and, for all $i = 1, \ldots, k$, a length-L substring s_i' of s_i such that $d_H(s, s_i') \leq d$?

Here $d_H(s, s_i')$ denotes the Hamming distance between s and s_i', i. e., the number of positions in which s and s_i' differ. Following the notation used in [7], m is used to denote the average length of the input strings and n to denote the total size of the problem input.

The optimization version of CLOSEST SUBSTRING asks for the minimum value of the distance parameter d for which the input strings still allow a solution.

Key Results

The classical complexity of CLOSEST SUBSTRING is given by

Theorem 1 ([4,5]) CLOSEST SUBSTRING *is NP-complete, and remains so for the special case of the* CLOSEST STRING *problem, where the requested solution string s has to be of same length as the input strings.* CLOSEST STRING *is NP-complete even for the further restriction to a binary alphabet.*

The following theorem gives the central statement concerning the problem's approximability:

Theorem 2 ([6]) CLOSEST SUBSTRING *(as well as* CLOSEST STRING*) admit polynomial time approximation schemes (PTAS's), where the objective function is the minimum Hamming distance d.*

In its randomized version, the PTAS cited by Theorem 2 computes, with high probability, a solution with Hamming distance $(1+\epsilon)d_{\text{opt}}$ for an optimum value d_{opt} in $(k^2 m)^{O(\log|\Sigma|/\epsilon^4)}$ running time. With additional overhead, this randomized PTAS can be derandomized. A straightforward and efficient factor-2 approximation for CLOSEST STRING is obtained by trying all length-L substrings of one of the input strings.

The following two statements address the problem's parametrized complexity, with respect to both obvious problem parameters d and k:

Theorem 3 ([3]) CLOSEST SUBSTRING *is W[1]-hard with respect to the parameter k, even for binary alphabet.*

Theorem 4 ([7]) CLOSEST SUBSTRING *is W[1]-hard with respect to the parameter d, even for binary alphabet.*

For non-binary alphabet the statement of Theorem 3 has been shown independently by Evans et al. [2]. Theorems 3 and 4 show that an exact algorithm for CLOSEST SUBSTRING with polynomial running time is unlikely for a constant value of d as well as for a constant value of k, i. e. such an algorithm does not exist unless 3-SAT can be solved in subexponential time.

Theorem 4 also allows additional insights into the problem's approximability: In the PTAS for CLOSEST SUBSTRING, the exponent of the polynomial bounding the running time depends on the approximation factor. These are not "efficient" PTAS's (EPTAS's), i. e. PTAS's with a $f(\epsilon) \cdot n^c$ running time for some function f and some constant c, and therefore are probably not useful in practice. Theorem 4 implies that most likely the PTAS with the $n^{O(1/\epsilon^4)}$ running time presented in [6] cannot be improved to an EPTAS. More precisely, there is no $f(\epsilon) \cdot n^{o(\log 1/\epsilon)}$ time PTAS for CLOSEST SUBSTRING unless 3-SAT can be solved in subexponential time. Moreover, the proof of Theorem 4 also yields

Theorem 5 ([7]) *There are no $f(d,k) \cdot n^{o(\log d)}$ time and no $g(d,k) \cdot n^{o(\log\log k)}$ exact algorithms solving* CLOSEST SUBSTRING *for some functions f and g unless* 3-SAT *can be solved in subexponential time.*

For unbounded alphabet the bounds have been strengthened by showing that Closest Substring has no PTAS with running time $f(\epsilon) \cdot n^{o(1/\epsilon)}$ for any function f unless 3-SAT can be solved in subexponential time [10]. The following statements provide exact algorithms for CLOSEST SUBSTRING with small fixed values of d and k, matching the bounds given in Theorem 5:

Theorem 6 ([7]) CLOSEST SUBSTRING *can be solved in time $f(d) \cdot n^{O(\log d)}$ for some function f, where, more precisely, $f(d) = |\Sigma|^{d(\log d+2)}$.*

Theorem 7 ([7]) CLOSEST SUBSTRING *can be solved in time $g(d,k) \cdot n^{O(\log\log k)}$ for some function g, where, more precisely, $g(d,k) = (|\Sigma|d)^{O(kd)}$.*

With regard to problem parameter L, CLOSEST SUBSTRING can be trivially solved in $O(|\Sigma|^L \cdot n)$ time by trying all possible strings over alphabet Σ.

Applications

An application of CLOSEST SUBSTRING lies in the analysis of biological sequences. In motif discovery, a goal is to search "signals" common to a set of selected strings representing DNA or protein sequences. One way to represent these signals are approximately preserved substrings occurring in each of the input strings. Employing Hamming distance as a biologically meaningful distance measure results in the problem formulation of CLOSEST SUBSTRING.

For example, Sagot [9] studies motif discovery by solving CLOSEST SUBSTRING (and generalizations thereof) using suffix trees; this approach has a worst-case running time of $O(k^2 m \cdot L^d \cdot |\Sigma|^d)$. In the context of motif discovery, also heuristics applicable to CLOSEST SUBSTRING were proposed, e. g., Pevzner and Sze [8] present an algorithm called WINNOWER and Buhler and Tompa [1] use a technique called random projections.

Open Problems

It is open [7] whether the $n^{O(1/\epsilon^4)}$ running time of the approximation scheme presented in [6] can be improved to $n^{O(\log 1/\epsilon)}$, matching the bound derived from Theorem 4.

Cross References

The following problems are close relatives of CLOSEST SUBSTRING:

- ▶ Closest String is the special case of CLOSEST SUBSTRING, where the requested solution string s has to be of same length as the input strings.
- Distinguishing Substring Selection is the generalization of CLOSEST SUBSTRING, where a second set of input strings and an additional integer d' are given and where the requested solution string s has – in addition to the requirements posed by CLOSEST SUBSTRING – Hamming distance at least d' with every length-L substring from the second set of strings.
- Consensus Patterns is the problem obtained by replacing, in the definition of CLOSEST SUBSTRING, the maximum of Hamming distances by the sum of Hamming distances. The resulting modified question of CONSENSUS PATTERNS is: Is there a string s of length L with

$$\sum_{i=1,\ldots,m} d_H(s, s_i') \leq d?$$

CONSENSUS PATTERNS is the special case of SUBSTRING PARSIMONY in which the phylogenetic tree provided in the definition of SUBSTRING PARSIMONY is a star phylogeny.

Recommended Reading

1. Buhler, J., Tompa, M.: Finding motifs using random projections. J. Comput. Biol. **9**(2), 225–242 (2002)
2. Evans, P.A., Smith, A.D., Wareham, H.T.: On the complexity of finding common approximate substrings. Theor. Comput. Sci. **306**(1–3), 407–430 (2003)
3. Fellows, M.R., Gramm, J., Niedermeier, R.: On the parameterized intractability of motif search problems. Combinatorica **26**(2), 141–167 (2006)
4. Frances, M., Litman, A.: On covering problems of codes. Theor. Comput. Syst. **30**, 113–119 (1997)
5. Lanctot, J.K.: Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing String Search Problems. Inf. Comput. **185**, 41–55 (2003)
6. Li, M., Ma, B., Wang, L.: On the Closest String and Substring Problems. J. ACM **49**(2), 157–171 (2002)
7. Marx, D.: The Closest Substring problem with small distances. In: Proceedings of the 46th FOCS, pp 63–72. IEEE Press, (2005)
8. Pevzner, P.A., Sze, S.H.: Combinatorial approaches to finding subtle signals in DNA sequences. In: Proc. of 8th ISMB, pp. 269–278. AAAI Press, (2000)
9. Sagot, M.F.: Spelling approximate repeated or common motifs using a suffix tree. In: Proc. of the 3rd LATIN, vol. 1380 in LNCS, pp. 111–127. Springer (1998)
10. Wang, J., Huang, M., Cheng, J.: A Lower Bound on Approximation Algorithms for the Closest Substring Problem. In: Proceedings COCOA 2007, vol. 4616 in LNCS, pp. 291–300 (2007)

Clustering

- ▶ Local Search for K-medians and Facility Location
- ▶ Well Separated Pair Decomposition for Unit–Disk Graph

Color Coding

1995; Alon, Yuster, Zwick

NOGA ALON[1], RAPHAEL YUSTER[2], URI ZWICK[3]

[1] Department of Mathematics and Computer Science, Tel-Aviv University, Tel-Aviv, Israel

[2] Department of Mathematics, University of Haifa, Haifa, Israel

[3] Department of Mathematics and Computer Science, Tel-Aviv University, Tel-Aviv, Israel

Keywords and Synonyms

Finding small subgraphs within large graphs

Problem Definition

Color coding [2] is a novel method used for solving, in polynomial time, various subcases of the generally NP-Hard *subgraph isomorphism* problem. The input for the

subgraph isomorphism problem is an ordered pair of (possibly directed) graphs (G, H). The output is either a mapping showing that H is isomorphic to a (possibly induced) subgraph of G, or **false** if no such subgraph exists. The subgraph isomorphism problem includes, as special cases, the HAMILTON-PATH, CLIQUE, and INDEPENDENT SET problems, as well as many others. The problem is also interesting when H is *fixed*. The goal, in this case, is to design algorithms whose running times are significantly better than the running time of the naïve algorithm.

Method Description

The color coding method is a randomized method. The vertices of the graph $G = (V, E)$ in which a subgraph isomorphic to $H = (V_H, E_H)$ is sought are randomly colored by $k = |V_H|$ colors. If $|V_H| = O(\log |V|)$, then with a small probability, but only polynomially small (i. e., one over a polynomial), all the vertices of a subgraph of G which is isomorphic to H, if there is such a subgraph, will be colored by distinct colors. Such a subgraph is called *color coded*. The color coding method exploits the fact that, in many cases, it is easier to detect color coded subgraphs than uncolored ones.

Perhaps the simplest interesting subcases of the subgraph isomorphism problem are the following: Given a directed or undirected graph $G = (V, E)$ and a number k, does G contain a simple (directed) path of length k? Does G contain a simple (directed) cycle of length *exactly* k? The following describes a $2^{O(k)} \cdot |E|$ time algorithm that receives as input the graph $G = (V, E)$, a coloring $c\colon V \to \{1, \ldots, k\}$ and a vertex $s \in V$, and finds a colorful path of length $k-1$ that starts at s, if one exists. To find a colorful path of length $k-1$ in G that starts somewhere, just add a new vertex s' to V, color it with a new color 0 and connect it with edges to all the vertices of V. Now look for a colorful path of length k that starts at s'.

A colorful path of length $k-1$ that starts at some specified vertex s is found using a dynamic programming approach. Suppose one is already given, for each vertex $v \in V$, the possible sets of colors on colorful paths of length i that connect s and v. Note that there is no need to record all colorful paths connecting s and v. Instead, record the color sets appearing on such paths. For each vertex v there is a collection of at most $\binom{k}{i}$ color sets. Now, inspect every subset C that belongs to the collection of v, and every edge $(v, u) \in E$. If $c(u) \notin C$, add the set $C \cup \{c(u)\}$ to the collection of u that corresponds to colorful paths of length $i+1$. The graph G contains a colorful path of length $k-1$ with respect to the coloring c if and only if the final collection, that corresponding to paths of length $k-1$, of at least one vertex is non-empty. The number of operations performed by the algorithm outlined is at most $O(\sum_{i=0}^{k} i\binom{k}{i} \cdot |E|)$ which is clearly $O(k2^k \cdot |E|)$.

Derandomization

The randomized algorithms obtained using the color coding method are derandomized with only a small loss in efficiency. All that is needed to derandomize them is a family of colorings of $G = (V, E)$ so that every subset of k vertices of G is assigned distinct colors by at least one of these colorings. Such a family is also called a family of *perfect hash functions* from $\{1, 2, \ldots, |V|\}$ to $\{1, 2, \ldots, k\}$. Such a family is explicitly constructed by combining the methods of [1,9,12,16]. For a derandomization technique yielding a constant factor improvement see [5].

Key Results

Lemma 1 *Let $G = (V, E)$ be a directed or undirected graph and let $c\colon V \to \{1, \ldots, k\}$ be a coloring of its vertices with k colors. A colorful path of length $k-1$ in G, if one exists, can be found in $2^{O(k)} \cdot |E|$ worst-case time.*

Lemma 2 *Let $G = (V, E)$ be a directed or undirected graph and let $c\colon V \to \{1, \ldots, k\}$ be a coloring of its vertices with k colors. All pairs of vertices connected by colorful paths of length $k-1$ in G can be found in either $2^{O(k)} \cdot |V||E|$ or $2^{O(k)} \cdot |V|^{\omega}$ worst-case time (here $\omega < 2.376$ denotes the matrix multiplication exponent).*

Using the above lemmata the following results are obtained.

Theorem 3 *A simple directed or undirected path of length $k-1$ in a (directed or undirected) graph $G = (V, E)$ that contains such a path can be found in $2^{O(k)} \cdot |V|$ expected time in the undirected case and in $2^{O(k)} \cdot |E|$ expected time in the directed case.*

Theorem 4 *A simple directed or undirected cycle of size k in a (directed or undirected) graph $G = (V, E)$ that contains such a cycle can be found in either $2^{O(k)} \cdot |V||E|$ or $2^{O(k)} \cdot |V|^{\omega}$ expected time.*

A cycle of length k in minor-closed families of graphs can be found, using color coding, even faster (for planar graphs, a slightly faster algorithm appears in [6]).

Theorem 5 *Let C be a non-trivial minor-closed family of graphs and let $k \geq 3$ be a fixed integer. Then, there exists a randomized algorithm that given a graph $G = (V, E)$ from C, finds a C_k (a simple cycle of size k) in G, if one exists, in $O(|V|)$ expected time.*

As mentioned above, all these theorems can be derandomized at the price of a $\log |V|$ factor. The algorithms are also easily to parallelize.

Applications

The initial goal was to obtain efficient algorithms for finding simple paths and cycles in graphs. The color coding method turned out, however, to have a much wider range of applicability. The linear time (i. e., $2^{O(k)} \cdot |E|$ for directed graphs and $2^{O(k)} \cdot |V|$ for undirected graphs) bounds for simple paths apply in fact to any *forest* on k vertices. The $2^{O(k)} \cdot |V|^{\omega}$ bound for simple cycles applies in fact to any *series-parallel* graph on k vertices. More generally, if $G = (V, E)$ contains a subgraph isomorphic to a graph $H = (V_H, E_H)$ whose *tree-width* is at most t, then such a subgraph can be found in $2^{O(k)} \cdot |V|^{t+1}$ expected time, where $k = |V_H|$. This improves an algorithm of Plehn and Voigt [14] that has a running time of $k^{O(k)} \cdot |V|^{t+1}$. As a very special case, it follows that the LOG PATH problem is in P. This resolves in the affirmative a conjecture of Papadimitriou and Yannakakis [13]. The exponential dependence on k in the above bounds is probably unavoidable as the problem is NP-complete if k is part of the input.

The color coding method has been a fruitful method in the study of parametrized algorithms and parametrized complexity [7,8]. Recently, the method has found interesting applications in computational biology, specifically in detecting signaling pathways within protein interaction networks, see [10,17,18,19].

Open Problems

Several problems, listed below, remain open.

- Is there a polynomial time (deterministic or randomized) algorithm for deciding if a given graph $G = (V, E)$ contains a path of length, say, $\log^2 |V|$? (This is unlikely, as it will imply the existence of an algorithm that decides in time $2^{O(\sqrt{n})}$ whether a given graph on n vertices is Hamiltonian.)
- Can the $\log |V|$ factor appearing in the derandomization be omitted?
- Is the problem of deciding whether a given graph $G = (V, E)$ contains a triangle as difficult as the Boolean multiplication of two $|V| \times |V|$ matrices?

Experimental Results

Results of running the basic algorithm on biological data have been reported in [17,19].

Cross References

- ▶ Approximation Schemes for Planar Graph Problems
- ▶ Graph Isomorphism
- ▶ Treewidth of Graphs

Recommended Reading

1. Alon, N., Goldreich, O., Håstad, J., Peralta, R.: Simple constructions of almost k-wise independent random variables. Random Struct. Algorithms **3**(3), 289–304 (1992)
2. Alon, N., Yuster, R., Zwick, U.: Color coding. J. ACM **42**, 844–856 (1995)
3. Alon, N., Yuster, R., Zwick, U.: Finding and counting given length cycles. Algorithmica **17**(3), 209–223 (1997)
4. Björklund, A., Husfeldt, T.: Finding a path of superlogarithmic length. SIAM J. Comput. **32**(6), 1395–1402 (2003)
5. Chen, J., Lu, S., Sze, S., Zhang, F.: Improved algorithms for path, matching, and packing problems. Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 298–307 (2007)
6. Eppstein, D.: Subgraph isomorphism in planar graphs and related problems. J. Graph Algorithms Appl. **3**(3), 1–27 (1999)
7. Fellows, M.R.: New Directions and new challenges in algorithm design and complexity, parameterized. In: Lecture Notes in Computer Science, vol. 2748, p. 505–519 (2003)
8. Flum, J., Grohe, M.: The Parameterized complexity of counting problems. SIAM J. Comput. **33**(4), 892–922 (2004)
9. Fredman, M.L., J.Komlós, Szemerédi, E.: Storing a sparse table with $O(1)$ worst case access time. J. ACM **31**, 538–544 (1984)
10. Hüffner, F., Wernicke, S., Zichner, T.: Algorithm engineering for Color Coding to facilitate Signaling Pathway Detection. In: Proceedings of the 5th Asia-Pacific Bioinformatics Conference (APBC), pp. 277–286 (2007)
11. Monien, B.: How to find long paths efficiently. Ann. Discret. Math. **25**, 239–254 (1985)
12. Naor, J., Naor, M.: Small-bias probability spaces: efficient constructions and applications. SIAM J. Comput. Comput. **22**(4), 838–856 (1993)
13. Papadimitriou, C.H., Yannakakis, M.: On limited nondeterminism and the complexity of the V-C dimension. J. Comput. Syst. Sci. **53**(2), 161–170 (1996)
14. Plehn, J., Voigt, B.: Finding minimally weighted subgraphs. Lect. Notes Comput. Sci. **484**, 18–29 (1990)
15. Robertson, N., Seymour, P.: Graph minors. II. Algorithmic aspects of tree-width. J. Algorithms **7**, 309–322 (1986)
16. Schmidt, J.P., Siegel, A.: The spatial complexity of oblivious k-probe hash functions. SIAM J. Comput. **19**(5), 775–786 (1990)
17. Scott, J., Ideker, T., Karp, R.M., Sharan, R.: Efficient Algorithms for Detecting Signaling Pathways in Protein Interaction Networks. J. Comput. Biol. **13**(2), 133–144 (2006)
18. Sharan, R., Ideker, T.: Modeling cellular machinery through biological network comparison. Nat. Biotechnol. **24**, 427–433 (2006)
19. Shlomi, T., Segal, D., Ruppin, E., Sharan, R.: QPath: a method for querying pathways in a protein-protein interaction network. BMC Bioinform. **7**, 199 (2006)

Communication in Ad Hoc Mobile Networks Using Random Walks

2003; Chatzigiannakis, Nikoletseas, Spirakis

Ioannis Chatzigiannakis
Department of Computer Engineering and Informatics, University of Patras and Computer Technology Institute, Patras, Greece

Keywords and Synonyms

Disconnected ad hoc networks; Delay-tolerant networks; Message Ferrying; Message relays; Data mules; Sink mobility

Problem Definition

A mobile ad hoc network is a temporary dynamic interconnection network of wireless mobile nodes without any established infrastructure or centralized administration. A *basic communication problem*, in ad hoc mobile networks, is to send information from a *sender* node, A, to another designated *receiver* node, B. If mobile nodes A and B come within wireless range of each other, then they are able to communicate. However, if they do not, they can communicate if other network nodes of the network are willing to forward their packets. One way to solve this problem is the protocol of notifying every node that the sender A meets and provide it with *all the information* hoping that some of them will eventually meet the receiver B.

> Is there a more efficient technique (other than notifying every node that the sender meets, in the hope that some of them will then eventually meet the receiver) that will effectively solve the communication establishment problem without flooding the network and exhausting the battery and computational power of the nodes?

The problem of communication among mobile nodes is one of the most fundamental problems in ad hoc mobile networks and is at the core of many algorithms, such as for counting the number of nodes, electing a leader, data processing etc. For an exposition of several important problems in ad hoc mobile networks see [13]. The work of Chatzigiannakis, Nikoletseas and Spirakis [5] focuses on wireless mobile networks that are subject to highly dynamic structural changes created by mobility, channel fluctuations and device failures. These changes affect topological connectivity, occur with high frequency and may not be predictable in advance. Therefore, the environment where the nodes move (in three-dimensional space with possible obstacles) as well as the motion that the nodes perform are *input* to any distributed algorithm.

The Motion Space

The space of possible motions of the mobile nodes is combinatorially abstracted by a *motion-graph*, i. e. the detailed geometric characteristics of the motion are neglected. Each mobile node is assumed to have a transmission range represented by a sphere tr centered by itself. Any other node inside tr can receive any message broadcast by this node. This sphere is approximated by a cube tc with volume $\mathcal{V}(tc)$, where $\mathcal{V}(tc) < \mathcal{V}(tr)$. The size of tc can be chosen in such a way that its volume $\mathcal{V}(tc)$ is the maximum that preserves $\mathcal{V}(tc) < \mathcal{V}(tr)$, and if a mobile node inside tc broadcasts a message, this message is received by any other node in tc. Given that the mobile nodes are moving in the space S, S is divided into consecutive cubes of volume $\mathcal{V}(tc)$.

Definition 1 The motion graph $G(V, E)$, ($|V| = n, |E| = m$), which corresponds to a quantization of S is constructed in the following way: a vertex $u \in G$ represents a cube of volume $\mathcal{V}(tc)$ and an edge $(u, v) \in G$ exists if the corresponding cubes are adjacent.

The number of vertices n, actually approximates the ratio between the volume $\mathcal{V}(S)$ of space S, and the space occupied by the transmission range of a mobile node $\mathcal{V}(tr)$. In the extreme case where $\mathcal{V}(S) \approx \mathcal{V}(tr)$, the transmission range of the nodes approximates the space where they are moving and $n = 1$. Given the transmission range tr, n depends linearly on the volume of space S regardless of the choice of tc, and $n = O(V(S)/V(tr))$. The ratio $V(S)/V(tr)$ is the *relative motion space size* and is denoted by ρ. Since the edges of G represent neighboring polyhedra each vertex is connected with a constant number of neighbors, which yields that $m = \Theta(n)$. In this example where tc is a cube, G has maximum degree of six and $m \leq 6n$. Thus *motion graph* G is (usually) a *bounded degree graph* as it is derived from a regular graph of small degree by deleting parts of it corresponding to motion or communication obstacles. Let Δ be the maximum vertex degree of G.

The Motion of the Nodes-Adversaries

In the general case, the motions of the nodes are decided by an *oblivious adversary*: The adversary determines motion patterns in any possible way but independently of the distributed algorithm. In other words, the case where some of the nodes are deliberately trying to *maliciously affect* the protocol, e. g. avoid certain nodes, are excluded. This is

a pragmatic assumption usually followed by applications. Such kind of motion adversaries are called *restricted motion adversaries*.

For purposes of studying efficiency of distributed algorithms for ad hoc networks *on the average*, the motions of the nodes are modeled by *concurrent and independent random walks*. The assumption that the mobile nodes move randomly, either according to uniformly distributed changes in their directions and velocities or according to the random waypoint mobility model by picking random destinations, has been used extensively by other research.

Key Results

The key idea is to take advantage of the mobile nodes natural movement by exchanging information whenever mobile nodes meet incidentally. It is evident, however, that if the nodes are spread in remote areas and they do not move beyond these areas, there is no way for information to reach them, unless the protocol takes special care of such situations. The work of Chatzigiannakis, Nikoletseas and Spirakis [5] proposes the idea of forcing only a small subset of the deployed nodes to move as per the needs of the protocol; they call this subset of nodes the *support* of the network. Assuming the availability of such nodes, they are used to provide a simple, correct and efficient strategy for communication between any pair of nodes of the network that avoids message flooding.

Let k nodes be a predefined set of nodes that become the nodes of the support. These nodes move randomly and fast enough so that they visit in sufficiently short time the entire motion graph. When some node of the support is within transmission range of a sender, it notifies the sender that it may send its message(s). The messages are then stored "somewhere within the support structure". When a receiver comes within transmission range of a node of the support, the receiver is notified that a message is "waiting" for him and the message is then forwarded to the receiver.

Protocol 1 (The "Snake" Support Motion Coordination Protocol) Let $S_0, S_1, \ldots, S_{k-1}$ be the members of the support and let S_0 denote the leader node (possibly elected). The protocol forces S_0 to perform a random walk on the motion graph and each of the other nodes S_i execute the simple protocol "move where S_{i-1} was before". When S_0 is about to move, it sends a message to S_1 that states the new direction of movement. S_1 will change its direction as per instructions of S_0 and will propagate the message to S_2. In analogy, S_i will follow the orders of S_{i-1} after transmitting the new directions to S_{i+1}. Movement orders received by S_i are positioned in a queue Q_i for sequential processing. The very first move of S_i, $\forall i \in \{1, 2, \ldots, k-1\}$ is delayed by a δ period of time.

The purpose of the random walk of the head S_0 is to ensure a *cover*, within some finite time, of the whole graph G without knowledge and memory, other than local, of topology details. This memoryless motion also ensures fairness, low-overhead and inherent robustness to structural changes.

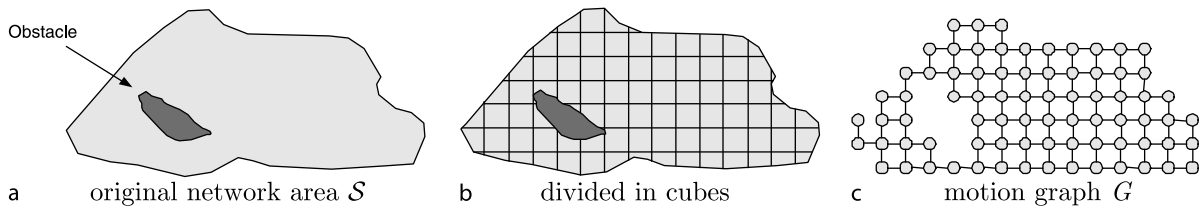
Consider the case where any sender or receiver is allowed a general, unknown motion strategy, but its strategy is provided by a restricted motion adversary. This means that each node not in the support either (a) executes a deterministic motion which either stops at a vertex or cycles forever after some initial part or (b) it executes a stochastic strategy which however is *independent* of the motion of the support. The authors in [5] prove the following correctness and efficiency results. The reader can refer to the excellent book by Aldous and Fill [1] for a nice introduction on Makrov Chains and Random Walks.

Theorem 1 *The support and the "snake" motion coordination protocol guarantee reliable communication between any sender-receiver (A, B) pair in finite time, whose expected value is bounded only by a function of the relative motion space size ρ and does not depend on the number of nodes, and is also independent of how MH_S, MH_R move, provided that the mobile nodes not in the support do not deliberately try to avoid the support.*

Theorem 2 *The expected communication time of the support and the "snake" motion coordination protocol is bounded above by $\Theta(\sqrt{mc})$ when the (optimal) support size $k = \sqrt{2mc}$ and c is $e/(e-1)u$, u being the "separation threshold time" of the random walk on G.*

Theorem 3 *By having the support's head move on a regular spanning subgraph of G, there is an absolute constant $\gamma > 0$ such that the expected meeting time of A (or B) and the support is bounded above by $\gamma n^2/k$. Thus the protocol guarantees a total expected communication time of $\Theta(\rho)$, independent of the total number of mobile nodes, and their movement.*

The analysis assumes that the head S_0 moves according to a continuous time random walk of total rate 1 (rate of exit out of a node of G). If S_0 moves ψ *times faster* than the rest of the nodes, all the estimated times, except the inter-support time, will be divided by ψ. Thus the expected total communication time can be made to be as small as $\Theta(\gamma\rho/\sqrt{\psi})$ where γ is an absolute constant. In cases where S_0 can take advantage of the network topology, all the estimated times, except the inter-support time are improved:



Communication in Ad Hoc Mobile Networks Using Random Walks, Figure 1
The original network area S (a), how it is divided in consecutive cubes of volume $\mathcal{V}(tc)$ (b) and the resulting motion graph G (c)

Theorem 4 *When the support's head moves on a regular spanning subgraph of G the expected meeting time of A (or B) and the support cannot be less than $(n-1)^2/2m$. Since $m = \Theta(n)$, the lower bound for the expected communication time is $\Theta(n)$. In this sense, the "snake" protocol's expected communication time is optimal, for a support size which is $\Theta(n)$.*

The "on-the-average" analysis of the time-efficiency of the protocol assumes that the motion of the mobile nodes not in the support *is a random walk on the motion graph G*. The random walk of each mobile node is performed independently of the other nodes.

Theorem 5 *The expected communication time of the support and the "snake" motion coordination protocol is bounded above by the formula*

$$E(T) \leq \frac{2}{\lambda_2(G)}\Theta\left(\frac{n}{k}\right) + \Theta(k)\,.$$

The upper bound is minimized when $k = \sqrt{2n/\lambda_2(G)}$, where λ_2 is the second eigenvalue of the motion graph's adjacency matrix.

The way the support nodes move and communicate is robust, in the sense that it can tolerate failures of the support nodes. The types of failures of nodes considered are permanent, i. e. stop failures. Once such a fault happens, the support node of the fault does not participate in the ad hoc mobile network anymore. A communication protocol is *β-faults tolerant*, if it still allows the members of the network to communicate correctly, under the presence of at most β permanent faults of the nodes in the support ($\beta \geq 1$). [5] shows that:

Theorem 6 *The support and the "snake" motion coordination protocol is 1-fault tolerant.*

Applications

Ad hoc mobile networks are rapidly deployable and self-configuring networks that have important applications in many critical areas such as disaster relief, ambient intelligence, wide area sensing and surveillance. The ability to network *anywhere*, *anytime* enables teleconferencing, home networking, sensor networks, personal area networks, and embedded computing applications [13].

Related Work

The most common way to establish communication is to form paths of intermediate nodes that lie within one another's transmission range and can directly communicate with each other. The mobile nodes act as hosts and routers at the same time in order to propagate packets along these paths. This approach of maintaining a global structure with respect to the temporary network is a difficult problem. Since nodes are moving, the underlying communication graph is changing, and the nodes have to adapt quickly to such changes and reestablish their routes. Busch and Tirthapura [2] provide the first analysis of the performance of some characteristic protocols [8,13] and show that in some cases they require $\Omega(u^2)$ time, where u is the number of nodes, to stabilize, i. e. be able to provide communication.

The work of Chatzigiannakis, Nikoletseas and Spirakis [5] focuses on networks where topological connectivity is subject to frequent, unpredictable change and studies the problem of efficient data delivery in sparse networks where network partitions can last for a significant period of time. In such cases, it is possible to have a small team of fast moving and versatile vehicles, to implement the support. These vehicles can be cars, motorcycles, helicopters or a collection of independently controlled mobile modules, i. e. robots. This specific approach is inspired by the work of Walter, Welch and Amato [14] that study the problem of motion co-ordination in distributed systems consisting of such robots, which can connect, disconnect and move around.

The use of mobility to improve performance in ad hoc mobile networks has been considered in different contexts in [6,9,11,15]. The primary objective has been to provide intermittent connectivity in a disconnected ad hoc net-

work. Each solution achieves certain properties of end-to-end connectivity, such as delay and message loss among the nodes of the network. Some of them require long-range wireless transmission, other require that all nodes move pro-actively under the control of the protocol and collaborate so that they meet more often. The *key idea* of forcing only a subset of the nodes to facilitate communication is used in a similar way in [10,15]. However, [15] focuses in cases where only one node is available. Recently, the application of mobility to the domain of wireless sensor networks has been addressed in [3,10,12].

Open Problems

A number of problems related to the work of Chatzigiannakis, Nikoletseas and Spirakis [5] remain open. It is clear that the size of the support, k, the shape and the way the support moves affects the performance of end-to-end connectivity. An open issue is to investigate alternative structures for the support, different motion coordination strategies and comparatively study the corresponding effects on communication times. To this end, the support idea is extended to hierarchical and highly changing motion graphs in [4]. The idea of cooperative routing based on the existence of support nodes may also improve security and trust.

An important issue for the case where the network is sparsely populated or where the rate of motion is too high is to study the performance of path construction and maintenance protocols. Some work has be done in this direction in [2] that can be also used to investigate the end-to-end communication in wireless sensor networks. It is still unknown if there exist impossibility results for distributed algorithms that attempt to maintain structural information of the implied fragile network of virtual links.

Another open research area is to analyze the properties of end-to-end communication given certain support motion strategies. There are cases where the mobile nodes interactions may behave in a similar way to the Physics paradigm of *interacting particles* and their modeling. Studies of interaction times and propagation times in various graphs are reported in [7] and are still important to further research in this direction.

Experimental Results

In [5] an experimental evaluation is conducted via simulation in order to model the different possible situations regarding the geographical area covered by an ad-hoc mobile network. A number of experiments were carried out for grid-graphs (2D, 3D), random graphs ($G_{n,p}$ model), bipartite multi-stage graphs and two-level motion graphs. All results verify the theoretical analysis and provide useful insight on how to further exploit the support idea. In [4] the model of hierarchical and highly changing ad-hoc networks is investigated. The experiments indicate that, the pattern of the "snake" algorithm's performance remains the same even in such type of networks.

URL to Code

http://ru1.cti.gr

Cross References

► Mobile Agents and Exploration

Recommended Reading

1. Aldous, D., Fill, J.: Reversible markov chains and random walks on graphs. http://stat-www.berkeley.edu/users/aldous/book.html (1999). Accessed 1999
2. Busch, C., Tirthapura, S.: Analysis of link reversal routing algorithms. SIAM J. Comput. 35(2):305–326 (2005)
3. Chatzigiannakis, I., Kinalis, A., Nikoletseas, S.: Sink mobility protocols for data collection in wireless sensor networks. In: Zomaya, A.Y., Bononi, L. (eds.) 4th International Mobility and Wireless Access Workshop (MOBIWAC 2006), Terromolinos, pp 52–59
4. Chatzigiannakis, I., Nikoletseas, S.: Design and analysis of an efficient communication strategy for hierarchical and highly changing ad-hoc mobile networks. J. Mobile Netw. Appl. **9**(4), 319–332 (2004). Special Issue on Parallel Processing Issues in Mobile Computing
5. Chatzigiannakis, I., Nikoletseas, S., Spirakis, P.: Distributed communication algorithms for ad hoc mobile networks. J. Parallel Distrib. Comput. (JPDC) **63**(1), 58–74 (2003). Special Issue on Wireless and Mobile Ad-hoc Networking and Computing, edited by Boukerche A
6. Diggavi, S.N., Grossglauser, M., Tse, D.N.C.: Even one-dimensional mobility increases the capacity of wireless networks. IEEE Trans. Inf. Theory **51**(11), 3947–3954 (2005)
7. Dimitriou, T., Nikoletseas, S.E., Spirakis, P.G.: Analysis of the information propagation time among mobile hosts. In: Nikolaidis, I., Barbeau, M., Kranakis, E. (eds.) 3rd International Conference on Ad-Hoc, Mobile, and Wireless Networks (ADHOC-NOW 2004), pp 122–134. Lecture Notes in Computer Science (LNCS), vol. 3158. Springer, Berlin (2004)
8. Gafni, E., Bertsekas, D.P.: Distributed algorithms for generating loop-free routes in networks with frequently changing topology. IEEE Trans. Commun. **29**(1), 11–18 (1981)
9. Grossglauser, M., Tse, D.N.C.: Mobility increases the capacity of ad hoc wireless networks. IEEE/ACM Trans. Netw. **10**(4), 477–486 (2002)
10. Jain, S., Shah, R., Brunette, W., Borriello, G., Roy, S.: Exploiting mobility for energy efficient data collection in wireless sensor networks. J. Mobile Netw. Appl. **11**(3), 327–339 (2006)
11. Li, Q., Rus, D.: Communication in disconnected ad hoc networks using message relay. Journal of Parallel and Distributed Computing (JPDC) **63**(1), 75–86 (2003). Special Issue on Wire-

less and Mobile Ad-hoc Networking and Computing, edited by A Boukerche
12. Luo, J., Panchard, J., Piórkowski, M., Grossglauser, M., Hubaux, J.P.: Mobiroute: Routing towards a mobile sink for improving lifetime in sensor networks. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) 2nd IEEE/ACM International Conference on Distributed Computing in Sensor Systems (DCOSS 2005). Lecture Notes in Computer Science (LNCS), vol. 4026, pp 480–497. Springer, Berlin (2006)
13. Perkins, C.E.: Ad Hoc Networking. Addison-Wesley, Boston (2001)
14. Walter, J.E., Welch, J.L., Amato, N.M.: Distributed reconfiguration of metamorphic robot chains. J. Distrib. Comput. **17**(2), 171–189 (2004)
15. Zhao, W., Ammar, M., Zegura, E.: A message ferrying approach for data delivery in sparse mobile ad hoc networks. In: Murai, J., Perkins, C., Tassiulas, L. (eds.) 5th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc 2004), pp 187–198. ACM Press, Roppongi Hills, Tokyo (2004)

Competitive Auction

2001; Goldberg, Hartline, Wright
2002; Fiat, Goldberg, Hartline, Karlin

Tian-Ming Bu
Department of Computer Science and Engineering, Fudan University, Shanghai, China

Problem Definition

This problem studies the *one round*, *sealed-bid* auction model where an auctioneer would like to sell an idiosyncratic commodity with unlimited copies to n bidders and each bidder $i \in \{1, \ldots, n\}$ will get at most one item.

First, for any i, bidder i bids a value b_i representing the price he is willing to pay for the item. They submit the bids simultaneously. After receiving the bidding vector $\mathbf{b} = (b_1, \ldots, b_n)$, the auctioneer computes and outputs the allocation vector $\mathbf{x} = (x_1, \ldots, x_n) \in \{0, 1\}^n$ and the price vector $\mathbf{p} = (p_1, \ldots, p_n)$. If for any i, $x_i = 1$, then bidder i gets the item and pays p_i for it. Otherwise, bidder i loses and pays nothing. In the auction, the auctioneer's revenue is $\sum_{i=1}^{n} \mathbf{x}\mathbf{p}^{\mathrm{T}}$.

Definition 1 (Optimal Single Price Omniscient Auction $\mathcal{F}$) Given a bidding vector $\mathbf{b}$ sorted in decreasing order,

$$\mathcal{F}(\mathbf{b}) = \max_{1 \le i \le n} i \cdot b_i.$$

Further,

$$\mathcal{F}^{(m)}(\mathbf{b}) = \max_{m \le i \le n} i \cdot b_i.$$

Obviously, $\mathcal{F}$ maximizes the auctioneer's revenue if only uniform price is allowed.

However, in this problem each bidder i is associated with a private value v_i representing the item's value in his opinion. So if bidder i gets the item, his payoff should be $v_i - p_i$. Otherwise, his payoff is 0. So for any bidder i, his payoff function can be formulated as $(v_i - p_i)x_i$. Furthermore, free will is allowed in the model. In other words, each bidder would bid some b_i different from his true value v_i, to maximize his payoff.

The objective of the problem is to design a *truthful* auction which could still maximize the auctioneer's revenue. An auction is *truthful* if for every bidder i, bidding his true value would maximize his payoff, regardless of the bids submitted by the other bidders [11,12].

Definition 2 (Competitive Auctions)
Input: the submitted bidding vector $\mathbf{b}$.
Output: the allocation vector $\mathbf{x}$ and the price vector $\mathbf{p}$.
Constraints:
(a) Truthful
(b) The auctioneer's revenue is within a constant factor of the optimal single pricing for all inputs.

Key Results

Let $\mathbf{b}_{-i} = (b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n)$. f is any function from $\mathbf{b}_{-i}$ to the price.

```
1: for i = 1 to n do
2:     if f(b_{-i}) ≤ b_i then
3:         x_i = 1 and p_i = f(b_i)
4:     else
5:         x_i = 0
6:     end if
7: end for
```

Competitive Auction, Algorithm 1
Bid-independent Auction: $\mathcal{A}_f(\mathbf{b})$

Theorem 1 ([6]) *An auction is truthful if and only if it is equivalent to a bid-independent auction.*

Definition 3 A truthful auction $\mathcal{A}$ is β-competitive against $\mathcal{F}^{(m)}$ if for all bidding vectors $\mathbf{b}$, the expected profit of $\mathcal{A}$ on $\mathbf{b}$ satisfies

$$\mathbf{E}(\mathcal{A}(\mathbf{b})) \ge \frac{\mathcal{F}^{(m)}(\mathbf{b})}{\beta}.$$

Definition 4 (CostShare$_C$) ([10]) Given bids $\mathbf{b}$, this mechanism finds the largest k such that the highest k bid-

ders' bids are at least C/k. Charge each of such k bidders C/k.

1: Partition bidding vector $\mathbf{b}$ uniformly at random into two sets $\mathbf{b}'$ and $\mathbf{b}''$.
2: Computer $\mathcal{F}' = \mathcal{F}(\mathbf{b}')$ and $\mathcal{F}'' = \mathcal{F}(\mathbf{b}'')$.
3: Running CostShare$_{\mathcal{F}''}$ on $\mathbf{b}'$ and CostShare$_{\mathcal{F}'}$ on $\mathbf{b}''$.

Competitive Auction, Algorithm 2
Sampling Cost Sharing Auction (SCS)

Theorem 2 ([6]) *SCS is 4-competitive against $\mathcal{F}^{(2)}$, and the bound is tight.*

Theorem 3 ([9]) *Let $\mathcal{A}$ be any truthful randomized auction. There exists an input bidding vector $\mathbf{b}$ on which $\boldsymbol{E}(\mathcal{A}(\mathbf{b})) \leq \frac{\mathcal{F}^{(2)}(\mathbf{b})}{2.42}$.*

Applications

As the Internet becomes more popular, more and more auctions are beginning to appear. Further, the items on sale in the auctions vary from antiques, paintings to digital goods such as mp3, licenses and network resources. Truthful auctions can reduce the bidders' cost of investigating the competitors' strategies, since truthful auctions encourage bidders to bid their true values. On the other hand, competitive auctions can also guarantee the auctioneer's profit. So this problem is very practical and significant. Over the last two years, designing and analyzing competitive auctions under various auction models have become a hot topic [1,2,3,4,5,7,8].

Cross References

- ▶ CPU Time Pricing
- ▶ Multiple Unit Auctions with Budget Constraint

Recommended Reading

1. Abrams, Z.: Revenue maximization when bidders have budgets. In: Proceedings of the seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-06), Miami, FL, 22–26 January 2006, pp. 1074–1082. ACM Press, New York (2006)
2. Bar-Yossef, Z., Hildrum, K., Wu, F.: Incentive-compatible online auctions for digital goods. In: Proceedings of the 13th Annual ACM-SIAM Symposium On Discrete Mathematics (SODA-02), New York, 6–8 January 2002, pp. 964–970. ACM Press, New York (2002)
3. Borgs, C., Chayes, J.T., Immorlica, N., Mahdian, M., Saberi, A.: Multi-unit auctions with budget-constrained bidders. In: ACM Conference on Electronic Commerce (EC-05), 2005, pp. 44–51
4. Bu, T.-M., Qi, Q., Sun, A.W.: Unconditional competitive auctions with copy and budget constraints. In: Spirakis, P.G., Mavronicolas, M., Kontogiannis, S.C. (eds.) Internet and Network Economics, 2nd International Workshop, WINE 2006, Patras, Greece, 15–17 Dec 2006. Lecture Notes in Computer Science, vol. 4286, pp. 16–26. Springer, Berlin (2006)
5. Deshmukh, K., Goldberg, A.V., Hartline, J.D., Karlin, A.R.: Truthful and competitive double auctions. In: Möhring, R.H., Raman, R. (eds.) Algorithms–ESA 2002, 10th Annual European Symposium, Rome, Italy, 17–21 Sept 2002. Lecture Notes in Computer Science, vol. 2461, pp. 361–373. Springer, Berlin (2002)
6. Fiat, A., Goldberg, A.V., Hartline, J.D., Karlin, A.R.: Competitive generalized auctions. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC-02), New York, 19–21 May 2002, pp. 72–81. ACM Press, New York (2002)
7. Goldberg, A.V., Hartline, J.D.: Competitive auctions for multiple digital goods. In: auf der Heide, F.M. (ed.) Algorithms – ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, 28–31 Aug 2001. Lecture Notes in Computer Science, vol. 2161, pp. 416–427. Springer, Berlin (2001)
8. Goldberg, A.V. Hartline, J.D.: Envy-free auctions for digital goods. In: Proceedings of the 4th ACM Conference on Electronic Commerce (EC-03), New York, 9–12 June 2003, pp. 29–35. ACM Press, New York (2003)
9. Goldberg, A.V., Hartline, J.D., Wright, A.: Competitive auctions and digital goods. In: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-01), New York, 7–9 January 2001, pp. 735–744. ACM Press, New York (2001)
10. Moulin, H.: Incremental cost sharing: Characterization by coalition strategy-proofness. Social Choice and Welfare, **16**, 279–320 (1999)
11. Nisan, N.and Ronen, A.: Algorithmic mechanism design. In: Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC-99), New York, May 1999, pp. 129–140. Association for Computing Machinery, New York (1999)
12. Parkes, D.C.: Chapter 2: Iterative Combinatorial Auctions. Ph. D. thesis, University of Pennsylvania (2004)

Complexity of Bimatrix Nash Equilibria

2006; Chen, Deng

Xi Chen[1], Xiaotie Deng[2]
[1] Computer Science and Technology, Tsinghua University, Beijing, China
[2] Department of Computer Science, City University of Hong Kong, Hong Kong, China

Keywords and Synonyms

Two-player nash; Two-player game; Two-person game; Bimatrix game

Problem Definition

In the middle of the last century, Nash [8] studied general non-cooperative games and proved that there exists a set

of mixed strategies, now commonly referred to as a Nash equilibrium, one for each player, such that no player can benefit if it changes its own strategy unilaterally. Since the development of Nash's theorem, researchers have worked on how to compute Nash equilibria efficiently. Despite much effort in the last half century, no significant progress has been made on characterizing its algorithmic complexity, though both hardness results and algorithms have been developed for various modified versions.

An exciting breakthrough, which shows that computing Nash equilibria is possibly hard, was made by Daskalakis, Goldberg, and Papadimitriou [4], for games among four players or more. The problem was proven to be complete in **PPAD** (polynomial parity argument, directed version), a complexity class introduced by Papadimitriou in [9]. The work of [4] is based on the techniques developed in [6]. This hardness result was then improved to the three-player case by Chen and Deng [1], Daskalakis and Papadimitriou [5], independently, and with different proofs. Finally, Chen and Deng [2] proved that NASH, the problem of finding a Nash equilibrium in a bimatrix game (or two-player game), is **PPAD**-complete.

A bimatrix game is a non-cooperative game between two players in which the players have m and n choices of actions (or pure strategies), respectively. Such a game can be specified by two $m \times n$ matrices $\mathbf{A} = (a_{i,j})$ and $\mathbf{B} = (b_{i,j})$. If the first player chooses action i and the second player chooses action j, then their payoffs are $a_{i,j}$ and $b_{i,j}$, respectively. A mixed strategy of a player is a probability distribution over its choices. Let $\mathbb{P}^n$ denote the set of all probability vectors in $\mathbb{R}^n$, i. e., non-negative vectors whose entries sum to 1. Nash's equilibrium theorem on non-cooperative games, when specialized to bimatrix games, states that, for every bimatrix game $\mathcal{G} = (\mathbf{A}, \mathbf{B})$, there exists a pair of mixed strategies $(\mathbf{x}^* \in \mathbb{P}^m, \mathbf{y}^* \in \mathbb{P}^n)$, called a Nash equilibrium, such that for all $\mathbf{x} \in \mathbb{P}^m$ and $\mathbf{y} \in \mathbb{P}^n$,

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

Computationally, one might settle with an approximate Nash equilibrium. Let $\mathbf{A}_i$ denote the ith row vector of $\mathbf{A}$, and $\mathbf{B}_i$ denote the ith column vector of $\mathbf{B}$. An ϵ-well-supported Nash equilibrium of game $(\mathbf{A}, \mathbf{B})$ is a pair of mixed strategies $(\mathbf{x}^*, \mathbf{y}^*)$ such that,

$$\mathbf{A}_i\mathbf{y}^* > \mathbf{A}_j\mathbf{y}^* + \epsilon \Longrightarrow x_j^* = 0, \ \forall\, i, j : 1 \leq i, j \leq m;$$
$$(\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}_i > (\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}_j + \epsilon \Longrightarrow y_j^* = 0, \ \forall\, i, j : 1 \leq i, j \leq n.$$

Definition 1 (2-NASH and NASH) The input instance of problem 2-NASH is a pair $(\mathcal{G}, 0^k)$ where $\mathcal{G}$ is a bimatrix game, and the output is a 2^{-k}-well-supported Nash equilibrium of $\mathcal{G}$. The input of problem NASH is a bimatrix game $\mathcal{G}$ and the output is an exact Nash equilibrium of $\mathcal{G}$.

Key Results

A binary relation $R \subset \{0,1\}^* \times \{0,1\}^*$ is *polynomially balanced* if there exists a polynomial p such that for all pairs $(x, y) \in R$, $|y| \leq p(|x|)$. It is a *polynomial-time computable relation* if for each pair (x, y), one can decide whether or not $(x, y) \in R$ in time polynomial in $|x| + |y|$. The **NP** search problem Q_R specified by R is defined as follows: Given $x \in \{0,1\}^*$, if there exists y such that $(x, y) \in R$, return y, otherwise, return a special string "no".

Relation R is *total* if for every $x \in \{0,1\}^*$, there exists a y such that $(x, y) \in R$. Following [7], let **TFNP** denote the class of all **NP** search problems specified by total relations. A search problem $Q_{R_1} \in$ **TFNP** is *polynomial-time reducible* to problem $Q_{R_2} \in$ **TFNP** if there exists a pair of polynomial-time computable functions (f, g) such that for every x of R_1, if y satisfies that $(f(x), y) \in R_2$, then $(x, g(y)) \in R_1$. Furthermore, Q_{R_1} and Q_{R_2} are polynomial-time equivalent if Q_{R_2} is also reducible to Q_{R_1}.

The complexity class **PPAD** is a sub-class of **TFNP**, containing all the search problems which are polynomial-time reducible to:

Definition 2 (Problem LEAFD) The input instance of LEAFD is a pair $(M, 0^n)$ where M defines a polynomial-time Turing machine satisfying:

1. for every $v \in \{0,1\}^n$, $M(v)$ is an ordered pair (u_1, u_2) with $u_1, u_2 \in \{0,1\}^n \cup \{\text{"no"}\}$;
2. $M(0^n) = (\text{"no"}, 1^n)$ and the first component of $M(1^n)$ is 0^n.

This instance defines a directed graph $G = (V, E)$ with $V = \{0,1\}^n$. Edge $(u, v) \in E$ iff v is the second component of $M(u)$ and u is the first component of $M(v)$.

The output of problem LEAFD is a directed leaf of G other than 0^n. Here a vertex is called a *directed leaf* if its out-degree plus in-degree equals one.

A search problem in **PPAD** is said to be *complete* in **PPAD** (or **PPAD**-complete), if there exists a polynomial-time reduction from LEAFD to it.

Theorem ([2]) 2-Nash *and* Nash *are* **PPAD**-*complete.*

Applications

The concept of Nash equilibria has traditionally been one of the most influential tools in the study of many disciplines involved with strategies, such as political science

and economic theory. The rise of the Internet and the study of its anarchical environment have made the Nash equilibrium an indispensable part of computer science. Over the past decades, the computer science community have contributed a lot to the design of efficient algorithms for related problems. This sequence of results [1,2,3,4,5,6], for the first time, provide *some evidence* that the problem of finding a Nash equilibrium is possibly hard for **P**. These results are very important to the emerging discipline, Algorithmic Game Theory.

Open Problems

This sequence of works show that $(r+1)$-player games are polynomial-time reducible to r-player games for every $r \geq 2$, but the reduction is carried out by first reducing $(r+1)$-player games to a fixed point problem, and then further to r-player games. Is there a natural reduction that goes directly from $(r+1)$-player games to r-player games? Such a reduction could provide a better understanding for the behavior of multi-player games.

Although many people believe that **PPAD** is hard for **P**, there is no strong evidence for this belief or intuition. The natural open problem is: Can one rigorously prove that class **PPAD** is hard, under one of those generally believed assumptions in theoretical computer science, like "**NP** is not in **P**" or "one way function exists"? Such a result would be extremely important to both Computational Complexity Theory and Algorithmic Game Theory.

Cross References

- ▶ General Equilibrium
- ▶ Leontief Economy Equilibrium
- ▶ Non-approximability of Bimatrix Nash Equilibria

Recommended Reading

1. Chen, X., Deng, X.: 3-Nash is ppad-complete. ECCC, TR05–134 (2005)
2. Chen, X., Deng, X.: Settling the complexity of two-player Nash-equilibrium. In: FOCS'06, Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, 2006, pp. 261–272
3. Chen, X., Deng, X., Teng, S.H.: Computing Nash equilibria: approximation and smoothed complexity. In: FOCS'06, Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, 2006, pp. 603–612
4. Daskalakis, C., Goldberg, P.W. Papadimitriou, C.H.: The complexity of computing a Nash equilibrium. In: STOC'06, Proceedings of the 38th ACM Symposium on Theory of Computing, 2006, pp. 71–78
5. Daskalakis, C., Papadimitriou, C.H.: Three-player games are hard. ECCC, TR05–139 (2005)
6. Goldberg, P.W., Papadimitriou, C.H.: Reducibility among equilibrium problems. In: STOC'06, Proceedings of the 38th ACM Symposium on Theory of Computing, 2006, pp. 61–70
7. Megiddo, N., Papadimitriou, C.H.: On total functions, existence theorems and computational complexity. Theor. Comp. Sci. **81**, 317–324 (1991)
8. Nash, J.F.: Equilibrium point in n-person games. In: Proceedings of the National Academy of the USA, vol. 36, issue 1, pp. 48–49 (1950)
9. Papadimitriou, C.H.: On the complexity of the parity argument and other inefficient proofs of existence. J. Comp. Syst. Sci. **48**, 498–532 (1994)

Complexity of Core

2001; Fang, Zhu, Cai, Deng

QIZHI FANG
Department of Mathematics,
Ocean University of China,
Qingdao, China

Keywords and Synonyms

Balanced; Least-core

Problem Definition

The core is the most important solution concept in cooperative game theory, which is based on the coalition rationality condition: no subgroup of the players will do better if they break away from the joint decision of all players to form their own coalition. The principle behind this condition is very similar and can be seen as an extension to that of the Nash Equilibrium. The problem of determining the core of a cooperative game naturally brings in issues of algorithms and complexity. The work of Fang, Zhu, Cai, and Deng [4] discusses the computational complexity issues related to the cores of some cooperative game models, such as, flow games and Steiner tree games.

A cooperative game with side payments is given by the pair $(N, \boldsymbol{v})$, where $N = \{1, 2, \cdots, n\}$ is the player set and $\boldsymbol{v} : 2^N \to R$ is the characteristic function. For each coalition $S \subseteq N$, the value $v(S)$ is interpreted as the profit or cost achieved by the collective action of players in S without any assistance of players in $N \setminus S$. A game is called a profit (cost) game if $v(S)$ measures the profit (cost) achieved by the coalition S. Here, the definitions are only given for profit games, symmetric statements hold for cost games. A vector $\boldsymbol{x} = \{x_1, x_2, \cdots, x_n\}$ is called an imputation if it satisfies $\sum_{i \in N} x_i = v(N)$ and $\forall i \in N : \; x_i \geq v(\{i\})$. The core of the game $(N, \boldsymbol{v})$ is de-

fined as:

$$C(v) = \{\boldsymbol{x} \in R^n : \boldsymbol{x}(N) = v(N) \text{ and } \boldsymbol{x}(S) \geq \boldsymbol{v}(S),\ \forall S \subseteq N\},$$

where $x(S) = \sum_{i \in S} x_i$ for $S \subseteq N$. A game is called *balanced*, if its core is non-empty; and *totally balanced*, if every subgame (i. e., the game obtained by restricting the player set to a coalition and the characteristic function to the power set of that coalition) is balanced.

It is a challenge for the algorithmic study of the core, since there are an exponential number of constraints imposed on its definition. The following computational complexity questions have attracted much attention from researchers:

(1)*Testing balancedness:* Can it be tested in polynomial time whether a given instance of the game has a non-empty core?

(2)*Checking membership:* Can it be checked in polynomial time whether a given imputation belongs to the core?

(3)*Finding a core member:* Is it possible to find an imputation in the core in polynomial time?

In reality, however, there is an important case in which the characteristic function value of a coalition can usually be evaluated via a combinatorial optimization problem, subject to constraints of resources controlled by the players of this coalition. In such circumstances, the input size of a game is the same as that of the related optimization problem, which is usually polynomial in the number of players. Therefore, this class of games, called combinatorial optimization games, fits well into the framework of algorithm theory. Flow games and Steiner tree games discussed in Fang et al. [4] fall within this scope.

FLOW GAME Let $D = (V, E; \omega; s, t)$ be a directed flow network, where V is the vertex set, E is the arc set, $\omega : E \to R^+$ is the arc capacity function, s and t are the source and the sink of the network, respectively. Assume that each player controls one arc in the network. The value of a maximum flow can be viewed as the profit achieved by the players in cooperation. Then the flow game $\Gamma_f = (E, \boldsymbol{v})$ associated with the network D is defined as follows:

(i) The player set is E;
(ii) $\forall S \subseteq E$, $\boldsymbol{v}(S)$ is the value of a maximum flow from s to t in the subnetwork of D consisting only of arcs belonging to S.

In Kailai and Zemel [6] and Deng et al. [2], it was shown that the flow game is totally balanced and finding a core member can be done in polynomial time.

Problem 1 *(Checking membership for flow game)*
INSTANCE: A flow network $D = (V, E; \omega; s, t)$ and $\boldsymbol{x} : E \to R^+$.
QUESTION: Is it true that $\boldsymbol{x}(E) = \boldsymbol{v}(E)$ and $\boldsymbol{x}(S) \geq \boldsymbol{v}(S)$ for all subsets $S \subset E$?

STEINER TREE GAME Let $G = (V, E; \omega)$ be an edge-weighted graph with $V = \{v_0\} \cup N \cup M$, where $N, M \subseteq V \setminus \{v_0\}$ are disjoint. v_0 represents a central supplier, N represents the consumer set, M represents the switch set, and $\omega(e)$ denotes the cost of connecting the two endpoints of edge e directly. It is required to connect all the consumers in N to the central supplier v_0. The connection is not limited to using direct links between two consumers or a consumer and the central supplier, it may pass through some switches in M. The aim is to construct the cheapest connection and distribute the connection cost among the consumers fairly. Then the associated Steiner tree game $\Gamma_s = (N, \gamma)$ is defined as follows:

(i) The player set is N;
(ii) $\forall\ S \subseteq N$, $\gamma(S)$ is the weight of a minimum Steiner tree on G w.r.t. the set $S \cup \{v_0\}$, that is, $\gamma(S) = \min\{\sum_{e \in E_S} \omega(e) : T_S = (V_S, E_S)$ is a subtree of G with $V_S \supseteq S \cup \{v_0\}\}$.

Different from flow games, the core of a Steiner tree game may be empty. An example with an empty core was given in Megiddo [9].

Problem 2 *(Testing balancedness for a Steiner tree game)*
INSTANCE: An edge-weighted graph $G = (V, E; \omega)$ with $V = \{v_0\} \cup N \cup M$.
QUESTION: Does there exist a vector $\boldsymbol{x} : N \to R^+$ such that $\boldsymbol{x}(N) = \gamma(N)$ and $\boldsymbol{x}(S) \leq \gamma(S)$ for all subsets $S \subset N$?

Problem 3 *(Checking membership for a Steiner tree game)*
INSTANCE: An edge-weighted graph $G = (V, E; \omega)$ with $V = \{v_0\} \cup N \cup M$ and $\boldsymbol{x} : N \to R^+$.
QUESTION: Is it true that $\boldsymbol{x}(N) = \gamma(N)$ and $\boldsymbol{x}(S) \leq \gamma(S)$ for all subsets $S \subset N$?

Key Results

Theorem 1 *It is $\mathcal{NP}$-complete to show that, given a flow game $\Gamma_f = (E, \boldsymbol{v})$ defined on network $D = (V, E; \omega; s, t)$ and a vector $\boldsymbol{x} : E \to R^+$ with $\boldsymbol{x}(E) = \boldsymbol{v}(E)$, whether there exists a coalition $S \subset N$ such that $\boldsymbol{x}(S) < \boldsymbol{v}(S)$. That is, checking membership of the core for flow games is co-$\mathcal{NP}$-complete.*

The proof of Theorem 1 yields directly the same conclusion for linear production games. In Owen's linear production game [10], each player j ($j \in N$) is in possession

of an individual resource vector $\boldsymbol{b}^j$. For a coalition S of players, the profit obtained by S is the optimum value of the following linear program:

$$\max\{c^t y : Ay \leq \sum_{j \in S} \boldsymbol{b}^j,\ y \geq 0\}.$$

That is, the characteristic function value is what the coalition can achieve in the linear production model with the resources under their control. Owen showed that one imputation in the core can also be constructed through an optimal dual solution to the linear program which determines the value of N. However, there are in general some imputations in the core which cannot be obtained in this way.

Theorem 2 *Checking membership of the core for linear production games is co-$\mathcal{NP}$-complete.*

The problem of finding a minimum Steiner tree in a network is $\mathcal{NP}$-hard, therefore, in a Steiner tree game, the value $\gamma(S)$ of each coalition S may not be obtained in polynomial time. It implies that the complement problem of checking membership of the core for Steiner tree games may not be in $\mathcal{NP}$.

Theorem 3 *It is $\mathcal{NP}$-hard to show that, given a Steiner tree game $\Gamma_s = (N, \gamma)$ defined on network $G = (V, E; \omega)$ and a vector $\boldsymbol{x} : N \to R^+$ with $\boldsymbol{x}(N) = \gamma(N)$, whether there exists a coalition $S \subset N$ such that $\boldsymbol{x}(S) > \gamma(S)$. That is, checking membership of the core for Steiner tree games is $\mathcal{NP}$-hard.*

Theorem 4 *Testing balancedness for Steiner tree games is $\mathcal{NP}$-hard.*

Given a Steiner tree game $\Gamma_s = (N, \gamma)$ defined on network $G = (V, E; \omega)$ and a subset $S \subseteq N$, in the subgame (S, γ_S), the value $\gamma(S')$ $(S' \subseteq S)$ is the weight of a minimum Steiner tree of G w.r.t. the subset $S' \cup \{v_0\}$, where all the vertices in $N \setminus S$ are treated as switches but not consumers. It is further proved in Fang et al. [4] that determining whether a Steiner tree game is totally balanced is also $\mathcal{NP}$-hard. This is the first example of $\mathcal{NP}$-hardness for the totally balanced condition.

Theorem 5 *Testing total balancedness for Steiner tree games is $\mathcal{NP}$-hard.*

Applications

The computational complexity results on the cores of combinatorial optimization games have been as diverse as the corresponding combinatorial optimization problems. For example:

(1) In matching games [1], testing balancedness, checking membership, and finding a core member can all be done in polynomial time;

(2) In flow games and minimum-cost spanning tree games [3,4], although their cores are always non-empty and a core member can be found in polynomial time, the problem of checking membership is *co-$\mathcal{NP}$*-complete;

(3) In facility location games [5], the problem of testing balancedness is in general $\mathcal{NP}$-hard, however, given the information that the core is non-empty, both finding a core member and checking membership can be solved efficiently;

(4) In a game of sum of edge weight defined on a graph [2], all the problems of testing balancedness, checking membership, and finding a core member are $\mathcal{NP}$-hard.

To make the study of complexity and algorithms for cooperative games meaningful to corresponding application areas, it is suggested that computational complexity be taken as an important factor in considering rationality and fairness of a solution concept, in a way derived from the concept of bounded rationality [3,8]. That is, the players are not willing to spend super-polynomial time to search for the most suitable solution. In the case when the solutions of a game do not exist or are difficult to compute or check, it may not be simple to dismiss the problem as hopeless, especially when the game arises from important applications. Hence, various conceptual approaches are proposed to resolve this problem.

When the core of a game is empty, it motivates conditions ensuring non-emptiness of approximate cores. A natural way to approximate the core is the *least core*. Let $(N, \boldsymbol{v})$ be a profit cooperative game. Given a real number ε, the ε-core is defined to contain the allocations such that $\boldsymbol{x}(S) \geq \boldsymbol{v}(S) - \varepsilon$ for each non-empty proper subset S of N. The *least core* is the intersection of all non-empty ε-cores. Let ε^* be the minimum value of ε such that the ε-core is empty, then the least core is the same as the ε^*-core.

The concept of the least core poses new challenges in regard to algorithmic issues. The most natural problem is how to efficiently compute the value ε^* for a given cooperative game. The catch is that the computation of ε^* requires solving of a linear program with an exponential number of constrains. Though there are cases where this value can be computed in polynomial time [7], it is in general very hard. If the value of ε^* is considered to represent some subsidies given by the central authority to ensure the existence of the cooperation, then it is significant to give the approximate value of it even when its computation is $\mathcal{NP}$-hard.

Another possible approach is to interpret approximation as bounded rationality. For example, it would be interesting to know if there is any game with a property that for any $\varepsilon > 0$, checking membership in the ε-core can be done in polynomial time but it is $\mathcal{NP}$-hard to tell if an imputation is in the core. In such cases, the restoration of cooperation would be a result of bounded rationality. That is to say, the players would not care an extra gain or loss of ε as the expense of another order of degree of computational resources. This methodology may be further applied to other solution concepts.

Cross References

- ▶ General Equilibrium
- ▶ Nucleolus
- ▶ Routing

Recommended Reading

1. Deng, X., Ibaraki, T., Nagamochi, H.: Algorithmic Aspects of the Core of Combinatorial Optimization Games. Math. Oper. Res. **24**, 751–766 (1999)
2. Deng, X., Papadimitriou, C.: On the Complexity of Cooperative Game Solution Concepts. Math. Oper. Res. **19**, 257–266 (1994)
3. Faigle, U., Fekete, S., Hochstättler, W., Kern, W.: On the Complexity of Testing Membership in the Core of Min-Cost Spanning Tree Games. Int. J. Game. Theor. **26**, 361–366 (1997)
4. Fang, Q., Zhu, S., Cai, M., Deng, X.: Membership for core of LP games and other games. COCOON 2001 Lecture Notes in Computer Science, vol. 2108, pp 247–246. Springer-Verlag, Berlin Heidelberg (2001)
5. Goemans, M.X., Skutella, M.: Cooperative Facility Location Games. J. Algorithms **50**, 194–214 (2004)
6. Kalai, E., Zemel, E.: Generalized Network Problems Yielding Totally Balanced Games. Oper. Res. **30**, 998–1008 (1982)
7. Kern, W., Paulusma, D.: Matching Games: The Least Core and the Nucleolus. Math. Oper. Res. **28**, 294–308 (2003)
8. Megiddo, N.: Computational Complexity and the Game Theory Approach to Cost Allocation for a Tree. Math. Oper. Res. **3**, 189–196 (1978)
9. Megiddo, N.: Cost Allocation for Steiner Trees. Netw. **8**, 1–6 (1978)
10. Owen, G.: On the Core of Linear Production Games. Math. Program. **9**, 358–370 (1975)

Compressed Pattern Matching

2003; Kida, Matsumoto, Shibata, Takeda, Shinohara, Arikawa

MASAYUKI TAKEDA
Department of Informatics, Kyushu University, Fukuoka, Japan

Keywords and Synonyms

String matching over compressed text; Compressed string search

Problem Definition

Let $\mathbf{c}$ be a given compression algorithm, and let $\mathbf{c}(A)$ denote the result of $\mathbf{c}$ compressing a string A. Given a pattern string P and a compressed text string $\mathbf{c}(T)$, the *compressed pattern matching (CPM)* problem is to find all occurrences of P in T without decompressing T. The goal is to perform the task in less time compared with a decompression followed by a simple search, which takes $O(|P| + |T|)$ time (assuming $O(|T|)$ time is enough for decompression). A CPM algorithm is said to be *optimal* if it runs in $O(|P| + |\mathbf{c}(T)|)$ time. The CPM problem was first defined in the work of Amir and Benson [1], and many studies have been made over different compression formats.

Collage Systems

Collage systems are useful CPM-oriented abstractions of compression formats, introduced by Kida et al. [9]. Algorithms designed for collage systems can be implemented for many different compression formats. In the same paper they designed a general Knuth–Morris–Pratt (KMP) algorithm for collage systems. A general Boyer–Moore (BM) algorithm for collage systems was also designed by almost the same authors [18].

A *collage system* is a pair $\langle \mathcal{D}, S \rangle$ defined as follows. $\mathcal{D}$ is a sequence of assignments $X_1 = expr_1; X_2 = expr_2; \ldots; X_n = expr_n$, where, for each $k = 1, \ldots, n$, X_k is a variable and $expr_k$ is any of the form:

a for $a \in \Sigma \cup \{\varepsilon\}$, (primitive assignment)

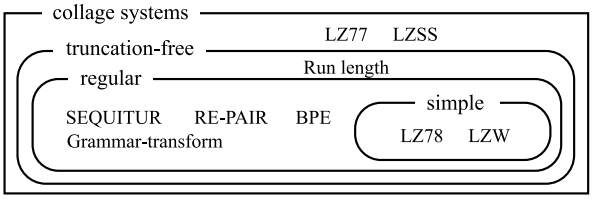
$X_i X_j$ for $i, j < k$, (concatenation)

${}^{[j]}X_i$ for $i < k$ and a positive integer j, (j length prefix truncation)

$X_i^{[j]}$ for $i < k$ and a positive integer j, (j length suffix truncation)

$(X_i)^j$ for $i < k$ and a positive integer j. (j times repetition)

By *the j length prefix (resp. suffix) truncation* we mean an operation on strings which takes a string w and returns the string obtained from w by removing its prefix (resp. suffix) of length j. The variables X_k represent the strings $\overline{X_k}$ obtained by evaluating their expressions. The *size* of $\mathcal{D}$ is the number n of assignments and denoted by $|\mathcal{D}|$. Let *height*$(\mathcal{D})$ denote the maximum dependence in $\mathcal{D}$. S is a sequence $X_{i_1} \cdots X_{i_\ell}$ of variables defined in $\mathcal{D}$. The *length*



Compressed Pattern Matching, Figure 1
Hierarchy of collage systems

of S is the number ℓ of variables in S and denoted by $|S|$. It can thus be considered that $|\mathbf{c}(T)| = |\mathcal{D}| + |S|$.

A collage system $\langle \mathcal{D}, S \rangle$ represents the string obtained by concatenating the strings $\overline{X_{i_1}}, \ldots, \overline{X_{i_\ell}}$ represented by variables $X_{i_1}, \ldots, X_{i_\ell}$ of S. It should be noted that any collage system can be converted into the one with $|S| = 1$, by adding a series of assignments with concatenation operations into $\mathcal{D}$. This may imply S is unnecessary. However, a variety of compression schemes can be captured naturally by separating $\mathcal{D}$ (defining *phrases*) from S (giving a factorization of text T into phrases). How to express compressed texts for existing compression schemes is found in [9].

A collage system is said to be *truncation-free* if $\mathcal{D}$ contains no truncation operation, and *regular* if $\mathcal{D}$ contains neither repetition nor truncation operation. A regular collage system is *simple* if $|\overline{Y}| = 1$ or $|\overline{Z}| = 1$ for every assignment $X = YZ$. Figure 1 gives the hierarchy of collage systems. The collage systems for RE-PAIR, SEQUITUR, Byte-Pair-Encoding (BPE), and the grammar-transform based compression scheme are regular. In the Lempel–Ziv family, the collage systems for LZ78/LZW are simple, while those for LZ77/LZSS are not truncation-free.

Key Results

It is straightforward to design an optimal solution for run-length encoding. For the two-dimensional run-length encoding, used by FAX transmission, an optimal solution was given by Amir, Benson, and Farach [3].

Theorem 1 (Amir et al. [3]) *There exists an optimal solution to the CPM problem for two-dimensional run-length encoding scheme.*

The same authors showed in [2] an almost optimal solution for LZW compression.

Theorem 2 (Amir et al. [2]) *The first-occurrence version of the CPM problem for LZW can be solved in* $O(|P|^2 + |\mathbf{c}(T)|)$ *time and space.*

An extension of [2] to the multi-pattern matching (dictionary matching) problem was presented by Kida et al. [10], together with the first experimental results in this area.

For LZ77 compression scheme, Farach and Thorup [6] presented the following result.

Theorem 3 (Farach and Thorup [6]) *Given an LZ77 compressed string Z of a text T, and given a pattern P, there is a randomized algorithm to decide if P occurs in T which runs in* $O(|Z| \log^2(|T|/|Z|) + |P|)$ *time.*

Lempel–Ziv factorization is a version of LZ77 compression without *self-referencing*. The following relation is present between Lempel–Ziv factorizations and collage systems.

Theorem 4 (Gąsieniec et al. [7]; Rytter [16]) *The Lempel–Ziv factorization Z of T can be transformed into a collage system of size* $O(|Z| \cdot \log |Z|)$ *generating T in* $O(|Z| \cdot \log |Z|)$ *time, and into a regular collage system of size* $O(|Z| \cdot \log |T|)$ *generating T in* $O(|Z| \cdot \log |T|)$ *time.*

The result of Amir et al. [2] was generalized in the work of Kida et al. [9] via the unified framework of collage systems.

Theorem 5 (Kida et al. [9]) *The CPM problem for collage systems can be solved in* $O\big((|\mathcal{D}|+|S|)\cdot height(\mathcal{D})+|P|^2+occ\big)$ *time using* $O(|\mathcal{D}| + |P|^2)$ *space, where occ is the number of pattern occurrences. The factor $height(\mathcal{D})$ is dropped for truncation-free collage systems.*

The algorithm of [9] has two stages: First it preprocesses $\mathcal{D}$ and P, and second it processes the variables of S. In the second stage, it simulates the move of a KMP automaton running on uncompressed text, by using two functions *Jump* and *Output*. Both these functions take a state q and a variable X as input. The former is used to substitute just one state transition for the consecutive state transitions of the KMP automaton for the string $\overline{X}$ for each variable X of S. The latter is used to report all pattern occurrences found during the state transitions. Let δ be the state-transition function of the KMP automaton. Then $Jump(q, X) = \delta(q, \overline{X})$ and $Output(q, X)$ is the set of lengths $|w|$ of non-empty prefixes w of $\overline{X}$ such that $\delta(q, w)$ is the final state. A naive two-dimensional array implementation of the two functions requires $\Omega(|\mathcal{D}| \cdot |P|)$ space. The data structures of [9] use only $O(|\mathcal{D}| + |P|^2)$ space, are built in $O(|\mathcal{D}| \cdot height(\mathcal{D}) + |P|^2)$ time, and enable us to compute $Jump(q, X)$ in $O(1)$ time and enumerate the set $Output(q, X)$ in $O(height(\mathcal{D}) + \ell)$ time where $\ell = |Output(q, X)|$. The factor $height(\mathcal{D})$ is dropped for truncation-free collage systems.

Another criterion of CPM algorithms is focused on the amount of extra space [4]. A CPM algorithm is *inplace* if

the amount of extra space is proportional to the input size of P.

Theorem 6 (Amir et al. [4]) *There exists an inplace CPM algorithm for a two-dimensional run-length encoding scheme which runs in $O(|\mathbf{c}(T)| + |P| \log \sigma)$ time using extra $O(\mathbf{c}(P))$ space, where σ is the minimum of $|P|$ and the alphabet size.*

Many variants of the CPM problem exist. In what follows, some of them are briefly sketched. *Fully-compressed pattern matching (FCPM)* is the complicated version where both T and P are given in a compressed format. A straight-line program is a regular collage system with $|S| = 1$.

Theorem 7 (Miyazaki et al. [13]) *The FCPM problem for straight-line programs is solved in $O(|\mathbf{c}(T)|^2 \cdot |\mathbf{c}(P)|^2)$ time using $O(|\mathbf{c}(T)| \cdot |\mathbf{c}(P)|)$ space.*

Approximate compressed pattern matching (ACPM) refers to the case where errors are allowed.

Theorem 8 (Kärkkäinen et al. [8]) *Under the Levenshtein distance model, the ACPM problem can be solved in $O(k \cdot |P| \cdot |\mathbf{c}(T)| + occ)$ time for LZ78/LZW, and in $O(|P| \cdot (k^2 \cdot |\mathcal{D}| + k \cdot |S|) + occ)$ time for regular collage systems, where k is the given error threshold.*

Theorem 9 (Makinen et al. [11]) *Under a weighted edit distance model, the ACPM problem for run-length encoding can be solved in $O(|P| \cdot |\mathbf{c}(P)| \cdot |\mathbf{c}(T)|)$ time.*

Regular expression compressed pattern matching (RCPM) refers to the case where P can be a regular expression.

Theorem 10 (Navarro [14]) *The RCPM problem can be solved in $O(2^{|P|} + |P| \cdot |\mathbf{c}(T)| + occ \cdot |P| \cdot \log |P|)$ time, where occ is the number of occurrences of P in T.*

Applications

CPM techniques enable us to search directly in compressed text databases. One interesting application is searching over compressed text databases on handheld devices, such as PDAs, in which memory, storage, and CPU power are limited.

Experimental Results

One important goal of the CPM problem is to perform a CPM task faster than a decompression followed by a simple search. Kida et al. [10] showed experimentally that their algorithms achieve the goal. Navarro and Tarhio [15] presented BM type algorithms for LZ78/LZW compression schemes, and showed they are twice as fast as a decompression followed by a search using the best algorithms. (The code is available at: www.dcc.uchile.cl/gnavarro/software.)

Another challenging goal is to perform a CPM task faster than a simple search over original files in the uncompressed format. The goal is achieved by Manber [12] (with his own compression scheme), and by Shibata et al. [17] (with BPE). Their search time reduction ratios are nearly the same as their compression ratios. Unfortunately the compression ratios are not very high. Moura et al. [5] achieved the goal by using a bytewise Huffman code on words. The compression ratio is relatively high, but only searching for whole words and phrases is allowed.

Cross References

► Multidimensional compressed pattern matching is the complex version of CPM where the text and the pattern are multidimensional strings in a compressed format. ► Sequential exact string matching, ► sequential approximate string matching, ► regular expression matching, respectively, refer to the simplified versions of CPM, ACPM, RCPM where the text and the pattern are given as uncompressed strings.

Recommended Reading

1. Amir, A., Benson, G.: Efficient two-dimensional compressed matching. In: Proc. Data Compression Conference '92 (DCC'92), pp. 279 (1992)
2. Amir, A., Benson, G., Farach, M.: Let sleeping files lie: Pattern matching in Z-compressed files. J. Comput. Syst. Sci. **52**(2), 299–307 (1996)
3. Amir, A., Benson, G., Farach, M.: Optimal two-dimensional compressed matching. J. Algorithms **24**(2), 354–379 (1997)
4. Amir, A., Landau, G.M., Sokol, D.: Inplace run-length 2d compressed search. Theor. Comput. Sci. **290**(3), 1361–1383 (2003)
5. de Moura, E., Navarro, G., Ziviani, N., Baeza-Yates, R.: Fast and flexible word searching on compressed text. ACM Trans. Inf. Syst. **18**(2), 113–139 (2000)
6. Farach, M., Thorup, M.: String-matching in Lempel–Ziv compressed strings. Algorithmica **20**(4), 388–404 (1998)
7. Gąsieniec, L., Karpinski, M., Plandowski, W., Rytter, W.: Efficient algorithms for Lempel–Ziv encoding. In: Proc. 5th Scandinavian Workshop on Algorithm Theory (SWAT'96). LNCS, vol. 1097, pp. 392–403 (1996)
8. Kärkkäinen, J., Navarro, G., Ukkonen, E.: Approximate string matching on Ziv–Lempel compressed text. J. Discret. Algorithms **1**(3–4), 313–338 (2003)
9. Kida, T., Matsumoto, T., Shibata, Y., Takeda, M., Shinohara, A., Arikawa, S.: Collage systems: a unifying framework for compressed pattern matching. Theor. Comput. Sci. **298**(1), 253–272 (2003)
10. Kida, T., Takeda, M., Shinohara, A., Miyazaki, M., Arikawa, S.: Multiple pattern matching in LZW compressed text. J. Discret. Algorithms **1**(1), 133–158 (2000)

11. Makinen, V., Navarro, G., Ukkonen, E.: Approximate matching of run-length compressed strings. Algorithmica **35**(4), 347–369 (2003)
12. Manber, U.: A text compression scheme that allows fast searching directly in the compressed file. ACM Trans. Inf. Syst. **15**(2), 124–136 (1997)
13. Miyazaki, M., Shinohara, A., Takeda, M.: An improved pattern matching algorithm for strings in terms of straight-line programs. J. Discret. Algorithms **1**(1), 187–204 (2000)
14. Navarro, G.: Regular expression searching on compressed text. J. Discret. Algorithms **1**(5–6), 423–443 (2003)
15. Navarro, G., Tarhio, J.: LZgrep: A Boyer–Moore string matching tool for Ziv–Lempel compressed text. Softw. Pract. Exp. **35**(12), 1107–1130 (2005)
16. Rytter, W.: Application of Lempel–Ziv factorization to the approximation of grammar-based compression. Theor. Comput. Sci. **302**(1–3), 211–222 (2003)
17. Shibata, Y., Kida, T., Fukamachi, S., Takeda, M., Shinohara, A., Shinohara, T., Arikawa, S.: Speeding up pattern matching by text compression. In: Proc. 4th Italian Conference on Algorithms and Complexity (CIAC'00). LNCS, vol. 1767, pp. 306–315. Springer, Heidelberg (2000)
18. Shibata, Y., Matsumoto, T., Takeda, M., Shinohara, A., Arikawa, S.: A Boyer–Moore type algorithm for compressed pattern matching. In: Proc. 11th Annual Symposium on Combinatorial Pattern Matching (CPM'00). LNCS, vol. 1848, pp. 181–194. Springer, Heidelberg (2000)

Compressed Suffix Array

2003; Grossi, Gupta, Vitter

Veli Mäkinen
Department of Computer Science,
University of Helsinki, Helsinki, Finland

Keywords and Synonyms

Compressed full-text indexing; Compressed suffix tree

Problem Definition

Given a *text string* $T = t_1 t_2 \ldots t_n$ over an alphabet Σ of size σ, the *compressed full-text indexing (CFTI)* problem asks to create a space-efficient data structure capable of efficiently simulating the functionalities of a *full-text index* build on T.

A simple example of a full-text index is *suffix array* $A[1, n]$ that contains a permutation of the interval $[1, n]$, such that $T[A[i], n] < T[A[i+1], n]$ for all $1 \leq i < n$, where "$<$" between strings is the lexicographical order. Using suffix array, the occurrences of a given pattern $P = p_1 p_2 \ldots p_m$ in T can be found using two binary searches in $O(m \log n)$ time.

The CFTI problem related to suffix arrays is easily stated; find a space-efficient data structure supporting the retrieval of value $A[i]$ for any i efficiently. Such a solution is called *compressed suffix array*. Usually compressed suffix arrays support, as well, retrieving of the inverse $A^{-1}[j] = i$ for any given j.

If the compressed full-text index functions without the text and contains enough information to retrieve any substring of T, then this index is called *self-index*, as it can be used as a representation of T. See the entry *Compressed Text Indexing* for another approach to self-indexing, and [9] for a comprehensive survey on the topic.

The CFTI problem can be stated on any full-text index, as long as the set of operations the data structure should support is rigorously defined. For example, a *compressed suffix tree* should simulate all the operations of classical *suffix trees*.

The classical full-text indexes occupy $O(n \log n)$ bits, typically with large constant factors. The typical goals in CFTI can be characterized by the degree of ambition; find a structure whose space-requirement is:

(i) proportional to the text size, i. e. $O(n \log \sigma)$ bits;
(ii) asymptotically optimal in the text size, i. e. $n \log \sigma (1 + o(1))$ bits;
(iii) proportional to the *compressed* text size, i. e. $O(nH_k)$ bits, where H_k is the (empirical) *k-th order entropy* of T[1]; or even
(iv) asymptotically optimal in the compressed text size, i. e. $nH_k + o(n \log \sigma)$ bits.

Key Results

The first solution to the problem is by Grossi and Vitter [3] who exploit the regularities of suffix array via the Ψ-function:

Definition 1 Given suffix array $A[1, n]$, *function* $\Psi : [1, n] \to [1, n]$ is defined so that, for all $1 \leq i \leq n$, $A[\Psi(i)] = A[i] + 1$. The exception is $A[1] = n$, in which case the requirement is that $A[\Psi(1)] = 1$ so that Ψ is a permutation.

Grossi and Vitter use a hierarchical decomposition of A based on Ψ[2]. Let us focus on the first level of that hierarchical decomposition. Let $A_0 = A$ be the original suffix array. A bit vector $B^0[1, n]$ is defined so that $B^0[i] = 1$ iff $A[i]$ is even. Let also $\Psi_0[1, \lceil n/2 \rceil]$ contain the sequence of values $\Psi(i)$ for arguments i where $B^0[i] = 0$. Finally, let $A_1[1, \lfloor n/2 \rfloor]$ be the subsequence of $A_0[1, n]$ formed by the even $A_0[i]$ values, divided by 2.

[1] H_k is the minimum average number of bits needed to code one symbol using any compressor that fixes the code word based on the k-symbol context following the the symbol to be coded. See [6] for more formal definition.

[2] The description below follows closely the one given in [9]

Then, $A = A_0$ can be represented using only Ψ_0, B^0, and A_1. To retrieve $A[i]$, first see if $B^0[i] = 1$. If it is, then $A[i]$ is (divided by 2) somewhere in A_1. The exact position depends on how many 1's are there in B^0 up to position i, denoted $rank(B^0, i)$; that is, $A[i] = 2 \cdot A_1[rank_1(B^0, i)]$. If $B^0[i] = 0$, then $A[i]$ is odd and not represented in A_1. However, $A[i] + 1 = A[\Psi(i)]$ has to be even and thus represented in A_1. Since Ψ_0 collects only the Ψ values where $B^0[i] = 0$, it holds that $A[\Psi(i)] = A[\Psi_0[rank_0(B^0, i)]]$. Once computing $A[\Psi(i)]$ (for even $\Psi(i)$), one simply obtains $A[i] = A[\Psi(i)] - 1$.

The idea can be used recursively: Instead of representing A_1, replace it with B^2, Ψ_2, and A_2. This is continued until A_h is small enough to be represented explicitly. The complexity is $O(h)$ assuming constant-time *rank*; one can attach $o(n)$ bits data structures to a bit vector of length n such that *rank*-queries can be be answered in constant time [4,7].

It is convenient to use $h = \lceil \log \log n \rceil$, so that the $n/2^h$ entries of A_h, each of which requires $O(\log n)$ bits, take overall $O(n)$ bits. All the B^ℓ arrays add up at most $2n$ bits (as their length is halved from each level to the next), and their additional *rank* structures add $o(n)$ extra bits. The only remaining problem is how to represent the Ψ_ℓ arrays. The following regularity due to lexicographic order can be exploited:

Lemma 1 *Given a text $T[1, n]$, its suffix array $A[1, n]$, and the corresponding function Ψ, it holds $\Psi(i) < \Psi(i + 1)$ whenever $T_{A[i]} = T_{A[i+1]}$.*

This piecewise increasing property of Ψ can be used to represent each level of Ψ in $\frac{1}{2} n \log \sigma$ bits [3]. Other tradeoffs are possible using different amount of levels:

Theorem 2 (Grossi and Vitter 2005 [3]) *The Compressed Suffix Array of Grossi and Vitter supports retrieving $A[i]$ in (i) $O(\log \log n)$ time using $n \log \sigma \log \log n + O(n \log \log \sigma)$ bits of space, or (ii) $O(\log^\epsilon n)$ time using $\frac{1}{\epsilon} n \log \sigma + O(n \log \log \sigma)$ bits of space, for any $0 < \epsilon < 1$.*

As a consequence, simulating the classical binary searches [5] to find the range of suffix array containing all the occurrences of a pattern $P[1, m]$ in $T[1, n]$, can then be done in $O(m \log^{1+\epsilon} n)$ time using space proportional to the text size. Reporting the *occ* occurrence positions takes $occ \times \log^\epsilon n$ time. This can be sped up when m is large enough [3].

Grossi and Vitter also show how to modify a space-efficient suffix tree [8] so as to obtain $O(m/\log_\sigma n + \log^\epsilon n)$ search time, for any constant $0 < \epsilon < 1$, using $O(n \log \sigma)$ bits of space.

Sadakane [10] shows how the above compressed suffix array can be converted into a self-index, and at the same time optimized in several ways. He does not give direct access to $A[i]$, but rather to any prefix of $T[A[i], n]$. This still suffices to use the binary search algorithm to locate the pattern occurrences.

Sadakane represents both A and T using the full function Ψ, and a few extra structures. Imagine one wishes to compare P against $T[A[i], n]$. For the binary search, one needs to extract enough characters from $T[A[i], n]$ so that its lexicographical relation to P is clear. Retrieving character $T[A[i]]$ is easy; Use a bit vector $F[1, n]$ marking the suffixes of $A[i]$ where the first character changes from that of $A[i - 1]$. After preprocessing F for *rank*-queries, computing $j = rank_1(F, i)$ tells us that $T[A[i]] = c_j$, where c_j is the j-th smallest alphabet character. Once $T[A[i]] = c_j$ is determined this way, one needs to obtain the next character, $T[A[i] + 1]$. But $T[A[i] + 1] = T[A[\Psi(i)]]$, so one can simply move to $i' = \Psi(i)$ and keep extracting characters with the same method, as long as necessary. Note that at most $|P| = m$ characters suffice to decide a comparison with P. Thus the binary search is simulated in $O(m \log n)$ time.

Up to now the space used is $n + o(n) + \sigma \log \sigma$ bits for F and Σ. Sadakane [10] gives an improved representation for Ψ using $nH_0 + O(n \log \log \sigma)$ bits, where H_0 is the zeroth order entropy of T.

Sadakane also shows how $A[i]$ can be retrieved, by plugging in the hierarchical scheme of Grossi and Vitter. He adds to the scheme the retrieval of the inverse $A^{-1}[j]$. This is used in order to retrieve arbitrary text substrings $T[p, r]$, by first applying $i = A^{-1}[p]$ and then continuing as before to retrieve $r - p + 1$ first characters of suffix $T[A[i], n]$. This capability turns the compressed suffix array into self-index:

Theorem 3 (Sadakane [10]) *The* Compressed Suffix Array *of Sadakane is a self-index occupying $\frac{1}{\epsilon} nH_0 + O(n \log \log \sigma)$ bits, and supporting retrieval of values $A[i]$ and $A^{-1}[j]$ in $O(\log^\epsilon n)$ time, counting of pattern occurrences in $O(m \log n)$ time, and displaying any substring of T of length ℓ in $O(\ell + \log^\epsilon n)$ time. Here $0 < \epsilon \leq 1$ is an arbitrary constant.*

Grossi, Gupta, and Vitter [1,2] have improved the space-requirement of compressed suffix arrays to depend on the k-th order entropy of T. The idea behind this improvement is a more careful analysis of regularities captured by the Ψ-function when combined with the indexing capabilities of their new elegant data structure, *wavelet tree*. They obtain, among other results, the following tradeoff:

Theorem 4 (Grossi, Gupta, and Vitter 2003 [2]) *The* Compressed Suffix Array *of Grossi, Gupta, and Vitter is a self-index occupying* $\frac{1}{\epsilon} nH_k + o(n \log \sigma)$ *bits, and supporting retrieval of values* $A[i]$ *and* $A^{-1}[j]$ *in* $O(\log^{1+\epsilon} n)$ *time, counting of pattern occurrences in* $O(m \log \sigma + \log^{2+\epsilon} n)$ *time, and displaying any substring of* T *of length* ℓ *in* $O(\ell / \log_\sigma n + \log^{1+\epsilon} n)$ *time. Here* $0 < \epsilon \leq 1$ *is an arbitrary constant,* $k \leq \alpha \log_\sigma n$ *for some constant* $0 < \alpha < 1$.

In the above, value k must be fixed before building the index. Later, they notice that a simple coding of Ψ-values yields the same nH_k bound without the need of fixing k beforehand [1].

Finally, compressed suffix arrays work as building blocks to solve other CFTI problems. For example, Sadakane [11] has created a fully functional compressed suffix tree by plugging in the compressed suffix array and the space-efficient suffix tree of Munro, Raman, and Rao [8]. This compressed suffix tree occupies $O(n \log \sigma)$ bits of space, simulating all suffix tree operations with at most $O(\log n)$ slowdown.

Applications

The application domains are the same as for the classical suffix arrays and trees, with the additional advantage of scaling up to significantly larger data sets.

URL to Code

See the corresponding *Compressed Text Indexing* entry for references to compressed suffix array implementations and http://www.cs.helsinki.fi/group/suds/cst for an implementation of Sadakane's compressed suffix tree.

Cross References

- ▶ Compressed Text Indexing
- ▶ Sequential Exact String Matching
- ▶ Text Indexing

Recommended Reading

1. Foschini, L., Grossi, R., Gupta, A., Vitter, J.S.: When indexing equals compression: Experiments with compressing suffix arrays and applications. ACM Trans. Algorithms **2**(4), 611–639 (2006)
2. Grossi, R., Gupta, A., Vitter, J.: High-order entropy-compressed text indexes. In: Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Baltimore, 12–14 January, pp. 841–850 (2003)
3. Grossi, R., Vitter, J.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching. SIAM J. Comput. **35**(2), 378–407 (2006)
4. Jacobson, G.: Space-efficient static trees and graphs. In: Proc. 30th IEEE Symposium on Foundations of Computer Science (FOCS), Research Triangle Park, 30 October – 1 November, pp. 549–554 (1989)
5. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. SIAM J. Comput. **22**(5), 935–948 (1993)
6. Manzini, G.: An analysis of the Burrows-Wheeler transform. J. ACM **48**(3), 407–430 (2001)
7. Munro, I.: Tables. In: Proc. 16th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS). LNCS, vol. 1180, Hyderabad, 18–20 December, pp. 37–42 (1996)
8. Munro, I., Raman, V., Rao, S.: Space efficient suffix trees. J. Algorithms **39**(2), 205–222 (2001)
9. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Comput. Surv. **39**(1), Article 2 (2007)
10. Sadakane, K.: New text indexing functionalities of the compressed suffix arrays. J. Algorithms **48**(2), 294–313 (2003)
11. Sadakane, K.: Compressed suffix trees with full functionality. Theor. Comput. Syst. **41**, 589–607 (2007)

Compressed Text Indexing

2005; Ferragina, Manzini

Veli Mäkinen[1], Gonzalo Navarro[2]

[1] Department of Computer Science, University of Helsinki, Helsinki, Finland

[2] Department of Computer Science, University of Chile, Santiago, Chile

Keywords and Synonyms

Space-efficient text indexing; Compressed full-text indexing; Self-indexing

Problem Definition

Given a *text string* $T = t_1 t_2 \ldots t_n$ over an alphabet Σ of size σ, the *compressed text indexing (CTI)* problem asks to *replace* T with a space-efficient data structure capable of efficiently answering basic string matching and substring queries on T. Typical queries required from such an index are the following:

- $count(P)$: count how many times a given *pattern string* $P = p_1 p_2 \ldots p_m$ occurs in T.
- $locate(P)$: return the locations where P occurs in T.
- $display(i, j)$: return $T[i, j]$.

Key Results

An elegant solution to the problem is obtained by exploiting the connection of *Burrows-Wheeler Transform (BWT)* [1] and *Suffix Array* data structure [9]. The suffix array $SA[1, n]$ of T is the permutation of text positions $(1 \ldots n)$ listing the *suffixes* $T[i, n]$ in lexicographic

order. That is, $T[SA[i], n]$ is the ith smallest suffix. The BWT is formed by (1) a permutation T^{bwt} of T defined as $T^{\text{bwt}}[i] = T[SA[i] - 1]$, where $T[0] = T[n]$, and (2) the number $i^* = SA^{-1}[1]$.

A property of the BWT is that symbols having the same context (i. e., string following them in T) are consecutive in T^{bwt}. This makes it easy to compress T^{bwt} achieving space close to high-order empirical entropies [10]. On the other hand, the suffix array is a versatile text index, allowing for example $O(m \log n)$ time counting queries (using two binary searches on SA) after which one can locate the occurrences in optimal time.

Ferragina and Manzini [3] discovered a way to combine the compressibility of the BWT and the indexing properties of the suffix array. The structure is essentially a compressed representation of the BWT plus some small additional structures to make it searchable.

We first focus on retrieving arbitrary substrings from this compressed text representation, and later consider searching capabilities. To retrieve the whole text from the structure (that is, to support $display(1, n)$), it is enough to invert the BWT. For this purpose, let us consider a table $LF[1, n]$ defined such that if $T[i]$ is permuted to $T^{\text{bwt}}[j]$ and $T[i-1]$ to $T^{\text{bwt}}[j']$ then $LF[j] = j'$. It is then immediate that T can be retrieved *backwards* by printing $T^{\text{bwt}}[i^*] \cdot T^{\text{bwt}}[LF[i^*]] \cdot T^{\text{bwt}}[LF[LF[i^*]]] \ldots$ (by definition $T^{\text{bwt}}[i^*]$ corresponds to $T[n]$).

To represent array LF space-efficiently, Ferragina and Manzini noticed that each $LF[i]$ can be expressed as follows:

Lemma 1 (Ferragina and Manzini 2005 [3]) *$LF[i] = C(c) + rank_c(i)$, where $c = T^{bwt}[i]$, $C(c)$ tells how many times symbols smaller than c appear in T^{bwt} and $rank_c(i)$ tells how many times symbol c appears in $T^{bwt}[1, i]$.*

General $display(i, j)$ queries rely on a regular sampling of the text. Every text position of the form $j' \cdot s$, being s the sampling rate, is stored together with $SA^{-1}[j' \cdot s]$, the suffix array position pointing to it. To solve $display(i, j)$ we start from the smallest sampled text position $j' \cdot s > j$ and apply the BWT inversion procedure starting with $SA^{-1}[j' \cdot s]$ instead of i^*. This gives the characters in reverse order from $j' \cdot s - 1$ to i, requiring at most $j - i + s$ steps.

It also happens that the very same two-part expression of $LF[i]$ enables efficient $count(P)$ queries. The idea is that if one knows the range of the suffix array, say $SA[sp_i, ep_i]$, such that the suffixes $T[SA[sp_i], n], T[SA[sp_i + 1], n], \ldots, T[SA[ep_i], n]$ are the only ones containing $P[i, m]$ as a prefix, then one can compute the new range $SA[sp_{i-1}, ep_{i-1}]$ where the suffixes contain $P[i-1, m]$ as a prefix, as follows: $sp_{i-1} = C(P[i-1]) + rank_{P[i-1]}(sp_i - 1) + 1$ and $ep_{i-1} = C(P[i-1]) + rank_{P[i-1]}(ep_i)$. It is then enough to scan the pattern *backwards* and compute values $C()$ and $rank_c()$ $2m$ times to find out the (possibly empty) range of the suffix array where all the suffixes start with the complete P. Returning $ep_1 - sp_1 + 1$ solves the $count(P)$ query without the need of having the suffix array available at all.

For locating each such occurrence $SA[i]$, $sp_1 \leq i \leq ep_1$, one can compute the sequence i, $LF[i]$, $LF[LF[i]]$, ..., until $LF^k[i]$ is a sampled suffix array position and thus it is explicitly stored in the sampling structure designed for $display(i, j)$ queries. Then $SA[i] = SA[LF^k[i]] + k$. As we are virtually moving sequentially on the text, we cannot do more than s steps in this process.

Now consider the space requirement. Values $C()$ can be stored trivially in a table of $\sigma \log_2 n$ bits. $T^{\text{bwt}}[i]$ can be computed in $O(\sigma)$ time by checking for which c is $rank_c(i) \neq rank_c(i-1)$. The sampling rate can be chosen as $s = \Theta(\log^{1+\epsilon} n)$ so that the samples require $o(n)$ bits. The only real challenge is to preprocess the text for $rank_c()$ queries. This has been a subject of intensive research in recent years and many solutions have been proposed. The original proposal builds several small partial sum data structures on top of the compressed BWT, and achieves the following result:

Theorem 2 (Ferragina and Manzini 2005 [3]) *The CTI problem can be solved using a so-called* FM-Index (FMI)*, of size $5nH_k + o(n \log \sigma)$ bits, that supports $count(P)$ in $O(m)$ time, $locate(P)$ in $O(\sigma \log^{1+\epsilon} n)$ time per occurrence, and $display(i, j)$ in $O(\sigma(j - i + \log^{1+\epsilon} n))$ time. Here H_k is the* kth order empirical entropy *of T, $\sigma = o(\log n / \log \log n)$, $k \leq \log_\sigma(n / \log n) - \omega(1)$, and $\epsilon > 0$ is an arbitrary constant.*

The original FM-Index has a severe restriction on the alphabet size. This has been removed in follow-up works. Conceptually, the easiest way to achieve a more alphabet-friendly instance of the FM-index is to build a *wavelet tree* [5] on T^{bwt}. This is a binary tree on Σ such that each node v handles a subset $S(v)$ of the alphabet, which is split among its children. The root handles Σ and each leaf handles a single symbol. Each node v encodes those positions i so that $T^{\text{bwt}}[i] \in S(v)$. For those positions, node v only stores a bit vector telling which go to the left, which to the right. The node bit vectors are preprocessed for constant time $rank_1()$ queries using $o(n)$-bit data structures [6, 12]. Grossi et al. [4] show that the wavelet tree built using the encoding of [12] occupies $nH_0 + o(n \log \sigma)$ bits. It is then easy to simulate a single $rank_c()$ query by $\log_2 \sigma$ $rank_1()$ queries. With the same cost one can obtain

$T^{\text{bwt}}[i]$. Some later enhancements have improved the time requirement, so as to obtain, for example, the following result:

Theorem 3 (Mäkinen and Navarro 2005 [7]) *The CTI problem can be solved using a so-called* Succinct Suffix Array (SSA), *of size* $nH_0 + o(n \log \sigma)$ *bits, that supports* $count(P)$ *in* $O(m(1 + \log \sigma / \log \log n))$ *time,* $locate(P)$ *in* $O(\log^{1+\epsilon} n \log \sigma / \log \log n)$ *time per occurrence, and* $display(i, j)$ *in* $O((j - i + \log^{1+\epsilon} n) \log \sigma / \log \log n)$ *time. Here* H_0 *is the zero-order entropy of* T, $\sigma = o(n)$, *and* $\epsilon > 0$ *is an arbitrary constant.*

Ferragina et al. [2] developed a technique called *compression boosting* that finds an optimal partitioning of T^{bwt} such that, when one compresses each piece separately using its zero-order model, the result is proportional to the kth order entropy. This can be combined with the idea of SSA by building a wavelet tree separately for each piece and some additional structures in order to solve global $rank_c()$ queries from the individual wavelet trees:

Theorem 4 (Ferragina et al. [4]) *The CTI problem can be solved using a so-called* Alphabet-Friendly FM-Index (AF-FMI), *of size* $nH_k + o(n \log \sigma)$ *bits, with the same time complexities and restrictions of SSA with* $k \leq \alpha \log_\sigma n$, *for any constant* $0 < \alpha < 1$.

A very recent analysis [8] reveals that the space of the plain SSA is bounded by the same $nH_k + o(n \log \sigma)$ bits, making the boosting approach to achieve the same result unnecessary in theory. In practice, implementations of [4, 7] are superior by far to those building directly on this simplifying idea.

Applications

Sequence analysis in Bioinformatics, search and retrieval on oriental and agglutinating languages, multimedia streams, and even structured and traditional database scenarios.

URL to Code and Data Sets

Site Pizza-Chili http://pizzachili.dcc.uchile.cl or http://pizzachili.di.unipi.it contains a collection of standardized library implementations as well as data sets and experimental comparisons.

Cross References

- ▶ Burrows–Wheeler Transform
- ▶ Compressed Suffix Array
- ▶ Sequential Exact String Matching
- ▶ Text Indexing

Recommended Reading

1. Burrows, M., Wheeler, D.: A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation (1994)
2. Ferragina, P., Giancarlo, R., Manzini, G., Sciortino, M.: Boosting textual compression in optimal linear time. J. ACM **52**(4), 688–713 (2005)
3. Ferragina, P. Manzini, G.: Indexing compressed texts. J. ACM **52**(4), 552–581 (2005)
4. Ferragina, P., Manzini, G., Mäkinen, V., Navarro, G.: Compressed representation of sequences and full-text indexes. ACM Trans. Algorithms **3**(2) Article 20 (2007)
5. Grossi, R., Gupta, A., Vitter, J.: High-order entropy-compressed text indexes. In: Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 841–850 (2003)
6. Jacobson, G.: Space-efficient static trees and graphs. In: Proc. 30th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 549–554 (1989)
7. Mäkinen, V., Navarro, G.: Succinct suffix arrays based on run-length encoding. Nord. J. Comput. **12**(1), 40–66 (2005)
8. Mäkinen, V., Navarro, G.: Dynamic entropy-compressed sequences and full-text indexes. In: Proc. 17th Annual Symposium on Combinatorial Pattern Matching (CPM). LNCS, vol. 4009, pp. 307–318 (2006) Extended version as TR/DCC-2006-10, Department of Computer Science, University of Chile, July 2006
9. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. SIAM J. Comput. **22**(5), 935–948 (1993)
10. Manzini, G.: An analysis of the Burrows-Wheeler transform. J. ACM **48**(3), 407–430 (2001)
11. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Comput. Surv. **39**(1) Article 2 (2007)
12. Raman, R., Raman, V., Rao, S.: Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In: Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 233–242 (2002)

Compressing Integer Sequences and Sets

2000; Moffat, Stuiver

Alistair Moffat
Department of Computer Science and Software Engineering, University of Melbourne, Melbourne, VIC, Australia

Problem Definition

Suppose that a message $M = \langle s_1, s_2, \ldots, s_n \rangle$ of length $n = |M|$ symbols is to be represented, where each symbol s_i is an integer in the range $1 \leq s_i \leq U$, for some upper limit U that may or may not be known, and may or may not be finite. Messages in this form are commonly the output of some kind of modeling step in a data compression system. The objective is to represent the message over a binary output alphabet $\{0, 1\}$ using as few as possible output

bits. A special case of the problem arises when the elements of the message are strictly increasing, $s_i < s_{i+1}$. In this case the message M can be thought of as identifying a subset of $\{1, 2, \ldots, U\}$. Examples include storing sets of IP addresses or product codes, and recording the destinations of hyperlinks in the graph representation of the world wide web.

A key restriction in this problem is that it may not be assumed that $n \gg U$. That is, it must be assumed that M is too short (relative to the universe U) to warrant the calculation of an M-specific code. Indeed, in the strictly increasing case, $n \leq U$ is guaranteed. A message used as an example below is $M_1 = \langle 1, 3, 1, 1, 1, 10, 8, 2, 1, 1\rangle$. Note that any message M can be converted to another message M' over the alphabet $U' = Un$ by taking prefix sums. The transformation is reversible, with the inverse operation known as "taking gaps".

Key Results

A key limit on static codes is expressed by the Kraft–McMillan inequality (see [13]): if the codeword for a symbol x is of length ℓ_x, then $\sum_{x=1}^{U} 2^{-\ell_x} \leq 1$ is required if the code is to be left-to-right decodeable, with no codeword a prefix of any other codeword. Another key bound is the combinatorial cost of describing a set. If an n-subset of $1 \ldots U$ is chosen at random, then a total of $\log_2 \binom{U}{n} \approx n \log_2(U/n)$ bits are required to describe that subset.

Unary and Binary Codes

As a first example method, consider *Unary coding*, in which the symbol x is represented as $x - 1$ bits that are `1`, followed by a single `0`-bit. For example, the first three symbols of message M_1 would be coded by "`0-110-0`", where the dashes are purely illustrative and do not form part of the coded representation. Because the Unary code for x is exactly x bits long, this code strongly favors small integers, and has a corresponding ideal symbol probability distribution (the distribution for which this particular pattern of codeword lengths yields the minimal message length) given by $Prob(x) = 2^{-x}$.

Unary has the useful attribute of being an infinite code. But unless the message M is dominated by small integers, Unary is a relatively expensive code. In particular, the Unary-coded representation of a message $M = \langle s_1 \ldots s_n\rangle$ requires $\sum_i s_i$ bits, and when M is a gapped representation of a subset of $1 \ldots U$, can be as long as U bits in total.

The best-known code in computing is *Binary*. If $2^{k-1} < U \leq 2^k$ for some integer k, then symbols $1 \leq s_i \leq U$ can be represented in $k \geq \log_2 U$ bits each. In this case, the code is *finite*, and the ideal probability distribution is given by $Prob(x) = 2^{-k}$. When $U = 2^k$, this then implies that $Prob(x) = 2^{-\log_2 n} = 1/n$.

When U is known precisely, and is not a power of two, $2^k - U$ of the codewords can be shortened to $k - 1$ bits long, in a *Minimal Binary* code. It is conventional to assign the short codewords to symbols $1 \cdots 2^k - U$. The codewords for the remaining symbols, $(2^k - U + 1) \cdots U$, remain k bits long.

Golomb Codes

In 1966 Solomon Golomb provided an elegant hybrid between Unary and Binary codes (see [15]). He observed that if a random n-subset of the items $1 \cdots U$ was selected, then the gaps between consecutive members of the subset were defined by a geometric probability distribution $Prob(x) = p(1 - p)^{x-1}$, where $p = n/U$ is the probability that any selected item is a member of the subset.

If b is chosen such that $(1 - p)^b = 0.5$, this probability distribution suggests that the codeword for $x + b$ should be one bit longer than the codeword for x. The solution $b = \log 0.5/\log(1 - p) \approx 0.69/p \approx 0.69U/n$ specifies a parameter b that defines the *Golomb code*. To then represent integer x, calculate $1 + ((x - 1) \operatorname{div} b)$ as a quotient, and code that part in Unary; and calculate $1 + ((x - 1) \bmod b)$ as a remainder part, and code it in Minimal Binary, against a maximum bound of b. When concatenated, the two parts form the codeword for integer x. As an example, suppose that $b = 5$ is specified. Then the five Minimal Binary codewords for the five possible binary suffix parts of the codewords are "`00`", "`01`", "`10`", "`110`", and "`111`". The number 8 is thus coded as a Unary prefix of "`10`" to indicate a quotient part of 2, followed by a Minimal Binary remainder of "`10`" representing 3, to make an overall codeword of "`10-10`".

Like Unary, the Golomb code is infinite; but by design is adjustable to different probability distributions. When $b = 2^k$ for integer k a special case of the Golomb code arises, usually called a *Rice code*.

Elias Codes

Peter Elias (again, see [15]) provided further hybrids between Unary and Binary codes in work published in 1975. This family of codes are defined recursively, with Unary being the simplest member.

To move from one member of the family to the next, the previous member is used to specify the number of bits in the standard binary representation of the value x being coded (that is, the value $1 + \lfloor \log_2 x \rfloor$); then, once the length

has been specified, the trailing bits of x, with the top bit suppressed, are coded in Binary.

For example, the second member of the Elias family is C_γ, and can be thought of as a Unary-Binary code: Unary to indicate the prefix part, being the magnitude of x; and then Binary to indicate the value of x within the range specified by the prefix part. The first few C_γ codewords are thus "0", "10-0", "10-1", "110-00", and so on, where the dashes are again purely illustrative. In general, the C_γ codeword for a value x requires $1 + \lfloor \log_2 x \rfloor$ bits for the Unary prefix part, and a further $\lfloor \log_2 x \rfloor$ for the binary suffix part, and the ideal probability distribution is thus given by $Prob(x) \geq 1/(2x^2)$.

After C_γ, the next member of the Elias family is C_δ. The only difference between C_γ codewords and the corresponding C_δ codewords is that in the latter C_γ is used to store the prefix part, rather than Unary. Further members of the family of Elias codes can be generated by applying the same process recursively, but for practical purposes C_δ is the last useful member of the family, even for relatively large values of x. To see why, note that $|C_\gamma(x)| \leq |C_\delta(x)|$ whenever $x \leq 31$, meaning that C_δ is longer than the next Elias code only for values $x \geq 2^{32}$.

Fibonacci-Based Codes

Another interesting code is derived from the Fibonacci sequence described (for this purpose) as $F_1 = 1$, $F_2 = 2$, $F_3 = 3$, $F_4 = 5$, $F_5 = 8$, and so on. The *Zeckendorf* representation of a natural number is a list of Fibonacci values that add up to that number, with the restriction that no two adjacent Fibonacci numbers may be used. For example, the number 10 is the sum of $2 + 8 = F_2 + F_5$.

The simplest *Fibonacci* code is derived directly from the ordered Zeckendorf representation of the target value, and consists of a "0" bit in the ith position (counting from the left) of the codeword if F_i does not appear in the sum, and a "1" bit in that position if it does, with indices considered in increasing order. Because it is not possible for both F_i and F_{i+1} to be part of the sum, the last two bits of this string must be "01". An appended "1" bit is thus sufficient to signal the end of each codeword. As always, the assumption of monotonically decreasing symbol probabilities means that short codes are assigned to small values. The code for integer one is "1-1", and the next few codewords are "01-1", "001-1", "101-1", "0001-1", "1001-1", where, as before, the embedded dash is purely illustrative.

Because $F_n \approx \phi^n$ where φ is the golden ratio $\phi = (1 + \sqrt{5})/2 \approx 1.61803$, the codeword for x is approximately $1 + \log_\phi x \approx 1 + 1.44 \log_2 x$ bits long, and is shorter than C_γ for all values except $x = 1$. It is also as good as, or better than, C_δ over a wide range of practical values between 2 and $F_{19} = 6{,}765$. Higher-order Fibonacci codes are also possible, with increased minimum codeword lengths, and decreased coefficients on the logarithmic term. Fenwick [8] provides good coverage of Fibonacci codes.

Byte Aligned Codes

Performing the necessary bit-packing and bit-unpacking operations to extract unrestricted bit sequences can be costly in terms of decoding throughput rates, and a whole class of codes that operate on units of bytes rather then bits have been developed – the *Byte Aligned* codes.

The simplest Byte Aligned code is an interleaved eight-bit analog of the Elias C_γ mechanism. The top bit in each byte is reserved for a flag that indicates (when "0") that "this is the last byte of this codeword" and (when "1") that "this is not the last byte of this codeword, take another one as well". The other seven bits in each byte are used for data bits. For example, the number 1,234 is coded into two bytes, "209-008", and is reconstructed via the calculation $(209 - 128 + 1) \times 128^0 + (008 + 1) \times 128^1 = 1,234$.

In this simplest byte aligned code, a total of $8\lceil (\log_2 x)/7 \rceil$ bits are used, which makes it more effective asymptotically than the $1 + 2\lfloor \log_2 x \rfloor$ bits required by the Elias C_γ code. However, the minimum codeword length of eight bits means that Byte Aligned codes are expensive on messages dominated by small values.

Byte Aligned codes are fast to decode. They also provide another useful feature – the facility to quickly "seek" forwards in the compressed stream over a given number of codewords. A third key advantage of byte codes is that if the compressed message is to be searched, the search pattern can be rendered into a sequence of bytes using the same code, and then any byte-based pattern matching utility be invoked [7]. The zero top bit in all final bytes means that false matches are identified with a single additional test.

An improvement to the simple Byte Aligned coding mechanism arises from the observation that there is nothing special about the value 128 as the separating value between the *stopper* and *continuer* bytes, and that different values lead to different tradeoffs in overall codeword lengths [3]. In these (S, C)-Byte Aligned codes, values of S and C such that $S + C = 256$ are chosen, and each codeword consists of a sequence of zero or more continuer bytes with values greater than or equal to S, and ends with a final stopper byte with a value less than S. Other variants include methods that use bytes as the cod-

ing units to form Huffman codes, either using eight-bit coding symbols or tagged seven-bit units [7]; and methods that partially permute the alphabet, but avoid the need for a complete mapping [6]. Culpepper and Moffat [6] also describe a byte aligned coding method that creates a set of byte-based codewords with the property that the first byte uniquely identifies the length of the codeword. Similarly, *Nibble* codes can be designed as a 4-bit analog of the Byte Aligned approach, where one bit is reserved for a stopper-continuer flag, and three bits are used for data.

Other Static Codes

There have been a wide range of other variants described in the literature. Several of these adjust the code by altering the boundaries of the set of *buckets* that define the code, and coding a value x as a Unary bucket identifier, followed by a Minimal Binary offset within the specified bucket (see [15]).

For example, the Elias C_γ code can be regarded as being a Unary-Binary combination relative to a vector of bucket sizes $\langle 2^0, 2^1, 2^2, 2^3, 2^4, \dots \rangle$. Teuhola (see [15]) proposed a hybrid in which a parameter k is chosen, and the vector of bucket sizes is given by $\langle 2^k, 2^{k+1}, 2^{k+2}, 2^{k+3}, \dots \rangle$. One way of setting the parameter k is to take it to be the length in bits of the median sequence value, so that the first bit of each codeword approximately halves the range of observed symbol values. Another variant method is described by Boldi and Vigna [2], who use a vector $\langle 2^k - 1, (2^k - 1)2^k, (2^k - 1)2^{2k}, (2^k - 1)2^{3k}, \dots \rangle$ to obtain a family of codes that are analytically and empirically well-suited to power-law probability distributions, especially those associated with web-graph compression. In this method k is typically in the range 2 to 4, and a Minimal Binary code is used for the suffix part.

Fenwick [8] provides detailed coverage of a wide range of static coding methods. Chen et al. [4] have also recently considered the problem of coding messages over sparse alphabets.

A Context Sensitive Code

The static codes described in the previous sections use the same set of codeword assignments throughout the encoding of the message. Better compression can be achieved in situations in which the symbol probability distribution is locally homogeneous, but not globally homogeneous.

Moffat and Stuiver [12] provided an off-line method that processes the message holisticly, in this case not because a parameter is computed (as is the case for the Binary code), but because the symbols are coded in a non-sequential manner. Their *Interpolative* code is a recursive coding method that is capable of achieving very compact representations, especially when the gaps are not independent of each other.

To explain the method, consider the subset form of the example message, as shown by sequence M_2 in Table 1. Suppose that the decoder is aware that the largest value in the subset does not exceed 29. Then every item in M is greater than or equal to $lo = 1$ and less than or equal to $hi = 29$, and the 29 different possibilities could be coded using Binary in fewer than $\lceil \log_2(29 - 1 + 1) \rceil = 5$ bits each. In particular, the mid-value in M_2, in this example the value $s_5 = 7$ (it doesn't matter which mid-value is chosen), can certainly be transmitted to the decoder using five bits. Then, once the middle number is pinned down, all of the remaining values can be coded within more precise ranges, and might require fewer than five bits each.

Now consider in more detail the range of values that the mid-value can span. Since there are $n = 10$ numbers in the list overall, there are four distinct values that precede s_5, and another five that follow it. From this argument a more restricted range for s_5 can be inferred: $lo' = lo + 4$ and $hi' = hi - 5$, meaning that the fifth value of M_2 (the number 7) can be Minimal Binary coded as a value within the range $[5, 24]$ using just 4 bits. The first row of Table 1 shows this process.

Now there are two recursive subproblems – transmitting the left part, $\langle 1, 4, 5, 6 \rangle$, against the knowledge that every value is greater than $lo = 1$ and $hi = 7 - 1 = 6$; and transmitting the right part, $\langle 17, 25, 27, 28, 29 \rangle$, against the knowledge that every value is greater than $lo = 7 + 1 = 8$ and less than or equal to $hi = 29$. These two sublists are processed recursively in the order shown in the remainder of Table 1, again with tighter ranges $[lo', hi']$ calculated and Minimal Binary codes emitted

One key aspect of the Interpolative code is that the situation can arise in which codewords that are zero bits long are called for, indicated when $lo' = hi'$. No bits need to be emitted in this case, since only one value is within the indicated range and the decoder can infer it. Four of the symbols in M_2 benefit from this possibility. This feature means that the Interpolative code is particularly effective when the subset contains clusters of consecutive items, or localized subset regions where there is a high density. In the limit, if the subset contains every element in the universal set, no bits at all are required once U is known. More generally, it is possible for dense sets to be represented in fewer than one bit per symbol.

Table 1 presents the Interpolative code using (in the final column) Minimal Binary for each value within its bounded range. A refinement is to use a Centered Minimal Binary code so that the short codewords are assigned

Compressing Integer Sequences and Sets, Table 1
Example encodings of message $M_2 = \langle 1, 4, 5, 6, 7, 17, 25, 27, 28, 29\rangle$ using the Interpolative code. When a Minimal Binary code is used, a total of 20 bits are required. When $lo' = hi'$, no bits are output

Index i	Value s_i	lo	hi	lo'	hi'	$\{s_i - lo', hi' - lo'\}$	Binary	MinBin
5	7	1	29	5	24	2,19	00010	0010
2	4	1	6	2	4	2,2	10	11
1	1	1	3	1	3	0,2	00	0
3	5	5	6	5	5	0,0	–	–
4	6	6	6	6	6	0,0	–	–
8	27	8	29	10	27	17,17	01111	11111
6	17	8	26	8	25	9,17	01001	1001
7	25	18	26	18	26	7,8	0111	1110
9	28	28	29	28	28	0,0	–	–
10	29	29	29	29	29	0,0	–	–

in the middle of the range rather than at the beginning, recognizing that the mid value in a set is more likely to be near the middle of the range spanned by those items than it is to the ends of the range. Adding this enhancement requires a trivial restructure of Minimal Binary coding, and tends to be beneficial in practice. But improvement is not guaranteed, and, as it turns out, on sequence M_2 the use of a Centered Minimal Binary code adds one bit to the length of the compressed representation compared to the Minimal Binary code shown in Table 1.

Cheng et al. [5] describe in detail techniques for fast decoding of Interpolative codes.

Hybrid Methods

It was noted above that the message must be assumed to be short relative to the total possible universe of symbols, and that $n \ll U$. Fraenkel and Klein [9] observed that the sequence of symbol *magnitudes* (that is, the sequence of values $\lceil \log_2 s_i \rceil$) in the message must be over a much more compact and dense range than the message itself, and it can be effective to use a principled code for the prefix parts that indicate the magnitudes, in conjunction with straightforward Binary codes for the suffix parts. That is, rather than using Unary for the prefix part, a Huffman (minimum-redundancy) code can be used.

In 1996 Peter Fenwick (see [13]) described a similar mechanism using Arithmetic coding, and as well incorporated an additional benefit. His *Structured Arithmetic* coder makes use of adaptive probability estimation and two-part codes, being a magnitude and a suffix part, with both calculated adaptively. The magnitude parts have a small range, and that code is allowed to adapt its inferred probability distribution quickly, to account for volatile local probability changes. The resultant two-stage coding process has the unique benefit of "smearing" probability changes across ranges of values, rather than confining them to the actual values recently processed.

Other Coding Methods

Other recent context sensitive codes include the *Binary Adaptive Sequential* code of Moffat and Anh [11]; and the *Packed Binary* codes of Anh and Moffat [1]. More generally, Witten et al. [15] and Moffat and Turpin [13] provide details of the Huffman and Arithmetic coding techniques that are likely to yield better compression when the length of the message M is large relative to the size of the source alphabet U.

Applications

A key application of compressed set representation techniques is to the storage of inverted indexes in large full-text retrieval systems of the kind operated by web search companies [15].

Open Problems

There has been recent work on compressed set representations that support operations such as *rank* and *select*, without requiring that the set be decompressed (see, for example, Gupta et al. [10] and Raman et al. [14]). Improvements to these methods, and balancing the requirements of effective compression versus efficient data access, are active areas of research.

Experimental Results

Comparisons based on typical data sets of a realistic size, reporting both compression effectiveness and decoding efficiency are the norm in this area of work. Witten et al.[15]

give details of actual compression performance, as do the majority of published papers.

URL to Code

The page at http://www.csse.unimelb.edu.au/~alistair/codes/ provides a simple text-based "compression" system that allows exploration of the various codes described here.

Cross References

- ► Arithmetic Coding for Data Compression
- ► Compressed Text Indexing
- ► Rank and Select Operations on Binary Strings

Recommended Reading

1. Anh, V.N., Moffat, A.: Improved word-aligned binary compression for text indexing. IEEE Trans. Knowl. Data Eng. **18**(6), 857–861 (2006)
2. Boldi, P., Vigna, S.: Codes for the world-wide web. Internet Math. **2**(4), 405–427 (2005)
3. Brisaboa, N.R., Fariña, A., Navarro, G., Esteller, M.F.: (*S*, *C*)-dense coding: An optimized compression code for natural language text databases. In: Nascimento, M.A. (ed.) Proc. Symp. String Processing and Information Retrieval. LNCS, vol. 2857, pp. 122–136, Manaus, Brazil, October 2003
4. Chen, D., Chiang, Y.J., Memon, N., Wu, X.: Optimal alphabet partitioning for semi-adaptive coding of sources of unknown sparse distributions. In: Storer, J.A., Cohn, M. (eds.) Proc. 2003 IEEE Data Compression Conference, pp. 372–381, IEEE Computer Society Press, Los Alamitos, California, March 2003
5. Cheng, C.S., Shann, J.J.J., Chung, C.P.: Unique-order interpolative coding for fast querying and space-efficient indexing in information retrieval systems. Inf. Process. Manag. **42**(2), 407–428 (2006)
6. Culpepper, J.S., Moffat, A.: Enhanced byte codes with restricted prefix properties. In: Consens, M.P., Navarro, G. (eds.) Proc. Symp. String Processing and Information Retrieval. LNCS Volume 3772, pp. 1–12, Buenos Aires, November 2005
7. de Moura, E.S., Navarro, G., Ziviani, N., Baeza-Yates, R.: Fast and flexible word searching on compressed text. ACM Trans. Inf. Syst. **18**(2), 113–139 (2000)
8. Fenwick, P.: Universal codes. In: Sayood, K. (ed.) Lossless Compression Handbook, pp. 55–78, Academic Press, Boston (2003)
9. Fraenkel, A.S., Klein, S.T.: Novel compression of sparse bit-strings –Preliminary report. In: Apostolico, A., Galil, Z. (eds) Combinatorial Algorithms on Words, NATO ASI Series F, vol. 12, pp. 169–183. Springer, Berlin (1985)
10. Gupta, A., Hon, W.K., Shah, R., Vitter, J.S.: Compressed data structures: Dictionaries and data-aware measures. In: Storer, J.A., Cohn, M. (eds) Proc. 16th IEEE Data Compression Conference, pp. 213–222, IEEE, Snowbird, Utah, March 2006 Computer Society, Los Alamitos, CA
11. Moffat, A., Anh, V.N.: Binary codes for locally homogeneous sequences. Inf. Process. Lett. **99**(5), 75–80 (2006) Source code available from www.cs.mu.oz.au/~alistair/rbuc/
12. Moffat, A., Stuiver, L.: Binary interpolative coding for effective index compression. Inf. Retr. **3**(1), 25–47 (2000)
13. Moffat, A., Turpin, A.: Compression and Coding Algorithms. Kluwer Academic Publishers, Boston (2002)
14. Raman, R., Raman, V., Srinivasa Rao, S.: Succinct indexable dictionaries with applications to encoding *k*-ary trees and multisets. In: Proc. 13th ACM-SIAM Symposium on Discrete Algorithms, pp. 233–242, San Francisco, CA, January 2002, SIAM, Philadelphia, PA
15. Witten, I.H., Moffat, A., Bell, T.C.: Managing Gigabytes: Compressing and Indexing Documents and Images, 2nd edn. Morgan Kaufmann, San Francisco, (1999)

Compression

- ► Compressed Suffix Array
- ► Compressed Text Indexing
- ► Rank and Select Operations on Binary Strings
- ► Similarity between Compressed Strings
- ► Table Compression

Computational Learning

- ► Learning Automata

Computing Pure Equilibria in the Game of Parallel Links

2002; Fotakis, Kontogiannis, Koutsoupias, Mavronicolas, Spirakis
2003; Even-Dar, Kesselman, Mansour
2003; Feldman, Gairing, Lücking, Monien, Rode

Spyros Kontogiannis
Department of Computer Science, University of Ioannina, Ioannina, Greece

Keywords and Synonyms

Load balancing game; Incentive compatible algorithms; Nashification; Convergence of Nash dynamics

Problem Definition

This problem concerns the construction of pure Nash equilibria (PNE) in a special class of atomic congestion games, known as the Parallel Links Game (PLG). The purpose of this note is to gather recent advances in the *existence and tractability of PNE in PLG*.

The Pure Parallel Links Game. Let $N \equiv [n]$[1] be a set of (selfish) players, each of them willing to have her

[1] $\forall k \in \mathbb{N}$, $[k] \equiv \{1, 2, \ldots, k\}$.

good served by a *unique* shared resource (link) of a system. Let $E = [m]$ be the set of all these links. For each link $e \in E$, and each player $i \in N$,let $D_{i,e}(\cdot) : \mathbb{R}_{\geq 0} \mapsto \mathbb{R}_{\geq 0}$ be the **charging mechanism** according to which link e charges player i for using it. Each player $i \in [n]$ comes with a service requirement (e. g. , traffic demand, or processing time) $W[i, e] > 0$, if she is to be served by link $e \in E$. A service requirement $W[i, e]$ is allowed to get the value ∞, to denote the fact that player i would never want to be assigned to link e. The charging mechanisms are functions of each link's cumulative congestion.

Any element $\sigma \in E$ is called a **pure strategy** for a player. Then, this player is assumed to assign her own good to link e. A collection of pure strategies for all the players is called a **pure strategies profile**, or a **configuration** of the players, or a **state** of the game.

The **individual cost** of player i wrt the profile σ is: $\mathtt{IC}_i(\sigma) = D_{i,\sigma_i}(\sum_{j\in[n]:\sigma_j=\sigma_i} W[j, \sigma_j])$. Thus, the **Pure Parallel Links Game** (PLG) is the game in strategic form defined as $\Gamma = \langle N, (\Sigma_i = E)_{i\in N}, (\mathtt{IC}_i)_{i\in N}\rangle$, whose acceptable solutions are only PNE. Clearly, an arbitrary instance of PLG can be described by the tuple $\langle N, E, (W[i, e])_{i\in N,e\in E}, (D_{i,e}(\cdot))_{i\in N,e\in E}\rangle$.

Dealing with Selfish Behavior. The dominant solution concept for finite games in strategic form, is the Nash Equlibrium [14]. The definition of pure Nash Equilibria for PLG is the following:

Definition 1 (Pure Nash Equilibrium) For any instance $\langle N, E, (W[i, e])_{i\in N,e\in E}, (D_{i,e}(\cdot))_{i\in N,e\in E}\rangle$ of PLG, a pure strategies profile $\sigma \in E^n$ is a **Pure Nash Equilibrium** (PNE in short), iff:
$\forall i \in N, \forall e \in E, \mathtt{IC}_i(\sigma) = D_{i,\sigma_i}\left(\sum_{j\in[n]:\sigma_j=\sigma_i} W[j, \sigma_i]\right)$
$\leq D_{i,e}\left(W[i, e] + \sum_{j\in[n]\setminus\{i\}:\sigma_j=e} W[j, e]\right)$.

A refinement of PNE are the ***k*-robust PNE**, for $n \geq k \geq 1$ [9]. These are pure profiles for which no subset of at most k players may concurrently change their strategies in such a way that the worst possible individual cost among the movers is *strictly decreased*.

Quality of Pure Equilibria. In order to determine the quality of a PNE, a social cost function that measures it must be specified. The typical assumption in the literature of PLG, is that the social cost function is the worst individual cost paid by the players: $\forall\sigma \in E^n$, $\mathtt{SC}(\sigma) = \max_{i\in N}\{\mathtt{IC}_i(\sigma)\}$ and $\forall \mathbf{p} \in (\Delta_m)^n$, $\mathtt{SC}(\mathbf{p}) = \sum_{\sigma\in E^n}(\prod_{i\in N} p_i(\sigma_i))\cdot\max_{i\in N}\{\mathtt{IC}_i(\sigma)\}$. Observe that, for mixed profiles, the social cost is the *expectation* of the maximum individual cost among the players.

The measure of the quality of an instance of PLG wrt PNE, is measured by the **Pure Price of Anarchy** (PPoA in short) [12]: $\mathtt{PPoA} = \max\{(\mathtt{SC}(\sigma))/\mathtt{OPT} : \sigma \in E^n \text{ is PNE}\}$ where $\mathtt{OPT} \equiv \min_{\sigma\in E^n}\{\mathtt{SC}(\sigma)\}$.

Discrete Dynamics. Crucial concepts of strategic games are the best and better responses. Given a configuration $\sigma \in E^n$, an **improvement step**, or **selfish step**, or **better response** of player $i \in N$ is the choice by i of a pure strategy $\alpha \in E \setminus \{\sigma_i\}$, so that player i would have a positive gain by this *unilateral* change (i. e., provided that the other players maintain the same strategies). That is, $\mathtt{IC}_i(\sigma) > \mathtt{IC}_i(\sigma \oplus_i \alpha)$ where, $\sigma \oplus_i \alpha \equiv (\sigma_1, \ldots, \sigma_{i-1}, \alpha, \sigma_{i+1}, \ldots, \sigma_n)$. A **best response**, or **greedy selfish step** of player i, is any change from the current link σ_i to an element $\alpha^* \in \arg\max_{a\in E}\{\mathtt{IC}_i(\sigma \oplus_i \alpha)\}$. An **improvement path** (aka a **sequence of selfish steps** [6], or an **elementary step system** [3]) is a sequence of configurations $\pi = \langle\sigma(1), \ldots, \sigma(k)\rangle$ such that

$$\forall 2 \leq r \leq k, \exists i_r \in N, \exists\alpha_r \in E:$$
$$[\sigma(r) = \sigma(r-1)\oplus_{i_r}\alpha_r]\wedge[\mathtt{IC}_{i_r}(\sigma(r)) < \mathtt{IC}_{i_r}(\sigma(r-1))].$$

A game has the **Finite Improvement Property** (FIP) iff any improvement path has *finite* length. A game has the **Finite Best Response Property** (FBRP) iff any improvement path, each step of whose is a best response of some player, has *finite* length.

An alternative trend is to, rather than consider *sequential* improvement paths, let the players conduct selfish improvement steps *concurrently*. Nevertheless, the selfish decisions are no longer deterministic, but rather distributions over the links, in order to have some notion of a priori Nash property that justifies these moves. The selfish players try to minimize their *expected* individual costs this time. Rounds of concurrent moves occur until a posteriori Nash Property is achieved. This is called a **selfish rerouting** policy [4].

Subclasses of PLG

[PLG$_1$] Monotone PLG: The charging mechanism of each pair of a link and a player, is a *non–decreasing function* of the resource's cumulative congestion.

[PLG$_2$] Resource Specific Weights PLG (RSPLG): Each player may have a different service demand from every link.

[PLG$_3$] Player Specific Delays PLG (PSPLG): Each link may have a different charging mechanism for each player. Some special cases of PSPLG are the following:

[PLG$_{3.1}$] Linear Delays PSPLG: Every link has a (player specific) *affine* charging mechanism: $\forall i \in N$, $\forall e \in E, D_{i,e}(x) = a_{i,e}x + b_{i,e}$ for some $a_{i,e} > 0$ and $b_{i,e} \geq 0$.

[**PLG**$_{3.1.1}$] **Related Delays PSPLG:** Every link has a (player specific) *non–uniformly related* charging mechanism: $\forall i \in N, \forall e \in E, W[i,e] = w_i$ and $D_{i,e}(x) = a_{i,e}x$ for some $a_{i,e} > 0$.

[**PLG**$_4$] **Resource Uniform Weights PLG (RUPLG):** Each player has a unique service demand from all the resources. Ie, $\forall i \in N, \forall e \in E, W[i,e] = w_e > 0$. A special case of RUPLG is:

[**PLG**$_{4.1}$] **Unweighted PLG**: All the players have identical demands from all the links: $\forall i \in N, \forall e \in E, W[i,e] = 1$.

[**PLG**$_5$] **Player Uniform Delays PLG (PUPLG):** Each resource adopts a unique charging mechanism, for all the players. That is, $\forall i \in N, \forall e \in E, D_{i,e}(x) = d_e(x)$.

[**PLG**$_{5.1}$] **Unrelated Parallel Machines**, or **Load Balancing PLG (LBPLG)**: The links behave as parallel machines. That is, they charge each of the players for the cumulative load assigned to their hosts. One may think (wlog) that all the machines have as charging mechanisms the identity function. That is, $\forall i \in N, \forall e \in E, D_{i,e}(x) = x$.

[**PLG**$_{5.1.1}$] **Uniformly Related Machines LBPLG:** Each player has the same demand at every link, and each link serves players at a fixed rate. That is: $\forall i \in N, \forall e \in E, W[i,e] = w_i$ and $D_{i,e}(x) = \frac{x}{s_e}$. Equivalently, service demands proportional to the capacities of the machines are allowed, but the identity function is required as the charging mechanism: $\forall i \in N, \forall e \in E, W[i,e] = \frac{w_i}{s_e}$ and $D_{i,e}(x) = x$.

[**PLG**$_{5.1.1.1}$] **Identical Machines LBPLG:** Each player has the same demand at every link, and all the delay mechanisms are the identity function: $\forall i \in N, \forall e \in E, W[i,e] = w_i$ and $D_{i,e}(x) = x$.

[**PLG**$_{5.1.2}$] **Restricted Assignment LBPLG:** Each traffic demand is either of unit or infinite size. The machines are identical. Ie, $\forall i \in N, \forall e \in E, W[i,e] \in \{1, \infty\}$ and $D_{i,e}(x) = x$.

Algorithmic Questions concerning PLG

The following algorithmic questions are considered:

Problem 1 (PNEExistsInPLG(E, N, W, D))
INPUT:
An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ *of PLG*
OUTPUT: *Is there a configuration* $\sigma \in E^n$ *of the players to the links, which is a PNE?*

Problem 2 (PNEConstructionInPLG(E, N, W, D))
INPUT:
An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ *of PLG*
OUTPUT: *An assignment* $\sigma \in E^n$ *of the players to the links, which is a PNE.*

Problem 3 (BestPNEInPLG(E, N, W, D))
INPUT:
An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ *of PLG. A* ***social cost*** *function* $SC : (\mathbb{R}_{\geq 0})^m \mapsto \mathbb{R}_{\geq 0}$ *that characterizes the quality of any configuration* $\sigma \in E^N$.
OUTPUT: *An assignment* $\sigma \in E^n$ *of the players to the links, which is a PNE and minimizes the value of the social cost, compared to other PNE of PLG.*

Problem 4 (WorstPNEInPLG(E, N, W, D))
INPUT:
An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ *of PLG. A* ***social cost*** *function* $SC : (\mathbb{R}_{\geq 0})^m \mapsto \mathbb{R}_{\geq 0}$ *that characterizes the quality of any configuration* $\sigma \in E^N$.
OUTPUT: *An assignment* $\sigma \in E^n$ *of the players to the links, which is a PNE and maximizes the value of the social cost, compared to other PNE of PLG.*

Problem 5 (DynamicsConvergeInPLG(E, N, W, D))
INPUT:
An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ *of PLG*
OUTPUT: *Does FIP (or FBRP) hold? How long does it take then to reach a PNE?*

Problem 6 (ReroutingConvergeInPLG(E, N, W, D))
INPUT:
An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ *of PLG*
OUTPUT: *Compute (if any) a selfish rerouting policy that converges to a PNE.*

Status of Problem 1

Player uniform, unweighted atomic congestion games always possess a PNE [15], with no assumption on monotonicity of the charging mechanisms. Thus, Problem 1 is already answered for all unweighted PUPLG. Nevertheless, this is not necessarily the case for weighted versions of PLG:

Theorem 1 ([13]) *There is an instance of (monotone) PSPLG with only three players and three strategies per player, possessing no PNE. On the other hand, any unweighted instance of monotone PSPLG possesses at least one PNE.*

Similar (positive) results were given for LBPLG. The key observation that lead to these results, is the fact that the lexicographically minimum vector of machine loads is always a PNE of the game.

Theorem 2 *There is always a PNE for any instance of Uniformly Related LBPLG* [7]*, and actually for any instance of LBPLG* [3]. Indeed, there is a $k-$robust PNE for any instance of LBPLG, and any $1 \leq k \leq n$ [9].

Status of Problems 2, 5 and 6

[13] gave a constructive proof of existence for PNE in unweighted, monotone PSPLG, and thus implies a path of length at most n that leads to a PNE. Although this is a very efficient construction of PNE, it is not necessarily an improvement path, when all players are considered to coexist all the time, and therefore there is no justification for the adoption of such a path by the players. Milchtaich [13] proved that from an arbitrary initial configuration and allowing only best reply defections, there is a best reply improvement path of length at most $m \cdot \binom{n+1}{2}$. Finally, [11] proved for unweighted, Related PSPLG that it possesses FIP. Nevertheless, the convergence time is poor.

For LBPLG, the implicit connection of PNE construction to classical scheduling problems, has lead to quite interesting results.

Theorem 3 ([7]) *The LPT algorithm of Graham, yields a PNE for the case of Uniformly Related LBPLG, in time* $\mathcal{O}(m \log m)$.

The drawback of the LPT algorithm is that it is centralized and not selfishly motivated. An alternative approach, called **Nashification**, is to start from an arbitrary initial configuration $\sigma \in E^n$ and then try to construct a PNE of at most the same maximum individual cost among the players.

Theorem 4 ([6]) *There is an* $\mathcal{O}(nm^2)$ *time Nashification algorithm for any instance of Uniformly Related PLG.*

An alternative style of Nashification, is to let the players follow an arbitrary improvement path. Nevertheless, it is not always the case that this leads to a polynomial time construction of a PNE, as the following theorem states:

Theorem 5 *For Identical Machines LBPLG:*

- *There exist best response improvement paths of length* $\Omega\left(\max\left\{2^{\sqrt{n}}, \left(\frac{n}{m^2}\right)^m\right\}\right)$ [3,6].
- *Any best response improvement path is of length* $\mathcal{O}(2^n)$ [6].
- *Any best response improvement path, which gives priority to players of maximum weight among those willing to defect in each improvement step, is of length at most n* [3].
- *If all the service demands are integers, then any improvement path which gives priority to unilateral improvement steps, and otherwise allows only selfish 2-flips (ie, swapping of hosting machines between two goods) converges to a 2-robust PNE in at most* $\frac{1}{2}(\sum_{i \in N} w_i)^2$ *steps* [9].

The following result concerns selfish rerouting policies:

Theorem 6 ([4])

- *For unweighted Identical Machines LBPLG, a simple policy (BALANCE) forcing all the players of overloaded links to migrate to a new (random) link with probability proportional to the load of the link, converges to a PNE in* $\mathcal{O}(\log \log n + \log m)$ *rounds of concurrent moves. The same convergence time holds also for a simple Nash Rerouting Policy, in which each mover actually has an incentive to move.*
- *For unweighted Uniformly Related LBPLG, BALANCE has the same convergence time, but the Nash Rerouting Policy may converge in* $\Omega\left(\sqrt{n}\right)$ *rounds.*

Finally, a generic result of [5] is mentioned, that computes a PNE for arbitrary *unweighted, player uniform* symmetric network congestion games in polynomial time, by a nice exploitation of Rosenthal's potential and the solution of a proper minimum cost flow problem. Therefore, for PLG the following result is implied:

Theorem 7 ([5]) *For unweighted, monotone PUPLG, a PNE can be constructed in polynomial time.*

Of course, this result provides no answer, e. g., for Restricted Assignment LBPLG, for which it is still not known how to efficiently compute PNE.

Status of Problems 3 and 4

The proposed LPT algorithm of [7] for constructing PNE in Uniformly Related LBPLG, actually provides a solution which is at most $1.52 < \text{PPoA}(LPT) < 1.67$ times worse than the optimum PNE (which is indeed the allocation of the goods to the links that minimizes the make-span). The construction of the optimum, as well as the worst PNE are hard problems, which nevertheless admits a PTAS (in some cases):

Theorem 8 *For LBPLG with a social cost function as defined in the* Quality of Pure Equilibria *paragraph:*

- *For Identical Machines, constructing the optimum or the worst PNE is* **NP***−hard* [7].
- *For Uniformly Related Machines, there is a PTAS for the optimum PNE* [6].

- *For Uniformly Related Machines, it holds that* $PPoA = \Theta\left(\min\{(\log m)/(\log\log m), \log(s_{\max})/(s_{\min})\}\right)$ [2].
- *For the Restricted Assignments,* $PPoA = \Omega((\log m)/(\log\log m))$ [10].
- *For a generalization of the Restricted Assignments, where the players have goods of any positive, otherwise infinite service demands from the links (and not only elements of* $\{1, \infty\}$*), it holds that* $m - 1 \leq PPoA < m$ [10].

It is finally mentioned that a recent result [1] for unweighted, single commodity network congestion games with linear delays, is translated to the following result for PLG:

Theorem 9 ([1]) *For unweighted PUPLG with linear charging mechanisms for the links, the worst case PNE may be a factor of* $PPoA = 5/2$ *away from the optimum solution, wrt the social cost defined in the* QUALITY OF PURE EQUILIBRIA *paragraph.*

Key Results

None

Applications

Congestion games in general have attracted much attention from many disciplines, partly because they capture a large class of routing and resource allocation scenarios.

PLG in particular, is the most elementary (non–trivial) atomic congestion game among a large number of players. Despite its simplicity, it was proved ([8] that it is asymptotically the worst case instance wrt the maximum individual cost measure, for a large class atomic congestion games involving the so called *layered networks*. Therefore, PLG is considered an excellent starting point for studying congestion games in large scale networks.

The importance of seeking for PNE, rather than arbitrary (mixed in general) NE, is quite obvious in sciences like the economics, ecology, and biology. It is also important for computer scientists, since it enforces deterministic costs to the players, and both the players and the network designer may feel safer in this case about what they will actually have to pay.

The question whether the Nash Dynamics converge to a PNE in a reasonable amount of time, is also quite important, since (in case of a positive answer) it justifies the selfish, decentralized, local dynamics that appear in large scale communications systems. Additionally, the selfish rerouting schemes are of great importance, since this is what should actually be expected from selfish, decentralized computing environments.

Open Problems

Open Question 1 *Determine the (in)existence of PNE for all the instances of PLG that do not belong in LBPLG, or in monotone PSPLG.*

Open Question 2 *Determine the (in)existence of k−robust PNE for all the instances of PLG that do not belong in LBPLG.*

Open Question 3 *Is there a polynomial time algorithm for constructing k−robust PNE, even for the Identical Machines LBPLG and* $k \geq 1$ *being a constant?*

Open Question 4 *Do the improvement paths of instances of PLG other than PSPLG and LBPLG converge to a PNE?*

Open Question 5 *Are there selfish rerouting policies of instances of PLG other than Identical Machines LBPLG converge to a PNE? How long much time would they need, in case of a positive answer?*

Cross References

- ► Best Response Algorithms for Selfish Routing
- ► Price of Anarchy
- ► Selfish Unsplittable Flows: Algorithms for Pure Equilibria

Recommended Reading

1. Christodoulou, G., Koutsoupias, E.: The Price of Anarchy of Finite Congestion Games. In: Proc. of the 37th ACM Symp. on Th. of Comp. (STOC '05), pp. 67–73. ACM, Baltimore (2005)
2. Czumaj, A., Vöcking, B.: Tight bounds for worst-case equilibria. In: Proc. of the 13th ACM-SIAM Symp. on Discr. Alg. (SODA '02), pp. 413–420. SIAM, San Francisco (2002)
3. Even-Dar, E., Kesselman, A., Mansour, Y.: Convergence time to nash equilibria. In: Proc. of the 30th Int. Col. on Aut., Lang. and Progr. (ICALP '03). LNCS, pp. 502–513. Springer, Eindhoven (2003)
4. Even-Dar, E., Mansour, Y.: Fast convergence of selfish rerouting. In: Proc. of the 16th ACM-SIAM Symp. on Discr. Alg. (SODA '05), SIAM, pp. 772–781. SIAM, Vancouver (2005)
5. Fabrikant, A., Papadimitriou, C., Talwar, K.: The complexity of pure nash equilibria. In: Proc. of the 36th ACM Symp. on Th. of Comp. (STOC '04). ACM, Chicago (2004)
6. Feldmann, R., Gairing, M., Lücking, T., Monien, B., Rode, M.: Nashification and the coordination ratio for a selfish routing game. In: Proc. of the 30th Int. Col. on Aut., Lang. and Progr. (ICALP '03). LNCS, pp. 514–526. Springer, Eindhoven (2003)
7. Fotakis, D., Kontogiannis, S., Koutsoupias, E., Mavronicolas, M., Spirakis, P.: The structure and complexity of nash equilibria

for a selfish routing game. In: Proc. of the 29th Int. Col. on Aut., Lang. and Progr. (ICALP '02). LNCS, pp. 123–134. Springer, Málaga (2002)
8. Fotakis, D., Kontogiannis, S., Spirakis, P.: Selfish unsplittable flows. Theor. Comput. Sci. **348**, 226–239 (2005) Special Issue dedicated to ICALP (2004) (TRACK-A)
9. Fotakis, D., Kontogiannis, S., Spirakis, P.: Atomic congestion games among coalitions. In: Proc. of the 33rd Int. Col. on Aut., Lang. and Progr. (ICALP '06). LNCS, vol. 4051, pp. 572–583. Springer, Venice (2006)
10. Gairing, M., Luecking, T., Mavronicolas, M., Monien, B.: The price of anarchy for restricted parallel links. Parallel Process. Lett. **16**, 117–131 (2006) Preliminary version appeared in STOC 2004
11. Gairing, M., Monien, B., Tiemann, K.: Routing (un-)splittable flow in games with player specific linear latency functions. In: Proc. of the 33rd Int. Col. on Aut., Lang. and Progr. (ICALP '06). LNCS, pp. 501–512. Springer, Venice (2006)
12. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Proc. of the 16th Annual Symp. on Theor. Aspects of Comp. Sci. (STACS '99), pp. 404–413. Springer, Trier (1999)
13. Milchtaich, I.: Congestion games with player-specific payoff functions. Games Econ. Behav. **13**, 111–124 (1996)
14. Nash, J.: Noncooperative games. Annals Math. **54**, 289–295 (1951)
15. Rosenthal, R.: A class of games possessing pure-strategy nash equilibria. Int. J. Game Theory **2**, 65–67 (1973)

Concurrent Programming, Mutual Exclusion

1965; Dijkstra

Gadi Taubenfeld
Department of Computer Science, Interdiciplinary Center Herzlia, Herzliya, Israel

Keywords and Synonyms

Critical section problem

Problem Definition

Concurrency, Synchronization and Resource Allocation

A *concurrent* system is a collection of processors that communicate by reading and writing from a shared memory. A distributed system is a collection of processors that communicate by sending messages over a communication network. Such systems are used for various reasons: to allow a large number of processors to solve a problem together much faster than any processor can do alone, to allow the distribution of data in several locations, to allow different processors to share resources such as data items, printers or discs, or simply to enable users to send electronic mail.

A *process* corresponds to a given computation. That is, given some program, its execution is a process. Sometimes, it is convenient to refer to the program code itself as a process. A process runs on a *processor*, which is the physical hardware. Several processes can run on the same processor although in such a case only one of them may be active at any given time. Real concurrency is achieved when several processes are running simultaneously on several processors.

Processes in a concurrent system often need to synchronize their actions. *Synchronization* between processes is classified as either cooperation or contention. A typical example for cooperation is the case in which there are two sets of processes, called the producers and the consumers, where the producers produce data items which the consumers then consume.

Contention arises when several processes compete for exclusive use of shared resources, such as data items, files, discs, printers, etc. For example, the integrity of the data may be destroyed if two processes update a common file at the same time, and as a result, deposits and withdrawals could be lost, confirmed reservations might have disappeared, etc. In such cases it is sometimes essential to allow at most one process to use a given resource at any given time.

Resource allocation is about interactions between processes that involve contention. The problem is, how to resolve conflicts resulting when several processes are trying to use shared resources. Put another way, how to allocate shared resources to competing processes. A special case of a general resource allocation problem is the *mutual exclusion* problem where only a single resource is available.

The Mutual Exclusion Problem

The *mutual exclusion* problem, which was first introduced by Edsger W. Dijkstra in 1965, is the guarantee of mutually exclusive access to a single shared resource when there are several competing processes [6]. The problem arises in operating systems, database systems, parallel supercomputers, and computer networks, where it is necessary to resolve conflicts resulting when several processes are trying to use shared resources. The problem is of great significance, since it lies at the heart of many interprocess synchronization problems.

The problem is formally defined as follows: it is assumed that each process is executing a sequence of instructions in an infinite loop. The instructions are divided into four continuous sections of code: the *remainder, entry, critical section* and *exit*. Thus, the structure of a mutual exclusion solution looks as follows:

loop forever
remainder code;
entry code;
critical section;
exit code
end loop

A process starts by executing the remainder code. At some point the process might need to execute some code in its critical section. In order to access its critical section a process has to go through an entry code which guarantees that while it is executing its critical section, no other process is allowed to execute its critical section. In addition, once a process finishes its critical section, the process executes its exit code in which it notifies other processes that it is no longer in its critical section. After executing the exit code the process returns to the remainder.

The Mutual exclusion problem is to write the code for the *entry code* and the *exit code* in such a way that the following two basic requirements are satisfied.

Mutual exclusion: *No two processes are in their critical sections at the same time.*

Deadlock-freedom: *If a process is trying to enter its critical section, then some process, not necessarily the same one, eventually enters its critical section.*
The deadlock-freedom property guarantees that the system as a whole can always continue to make progress. However deadlock-freedom may still allow "starvation" of individual processes. That is, a process that is trying to enter its critical section, may never get to enter its critical section, and wait forever in its entry code. A stronger requirement, which does not allow starvation, is defined as follows.

Starvation-freedom: *If a process is trying to enter its critical section, then this process must eventually enter its critical section.*
Although starvation-freedom is strictly stronger than deadlock-freedom, it still allows processes to execute their critical sections arbitrarily many times before some trying process can execute its critical section. Such a behavior is prevented by the following fairness requirement.

First-in-first-out (FIFO): *No beginning process can enter its critical section before a process that is already waiting for its turn to enter its critical section.*
The first two properties, mutual exclusion and deadlock freedom, were required in the original statement of the problem by Dijkstra. They are the minimal requirements that one might want to impose. In solving the problem, it is assumed that once a process starts executing its critical section the process always finishes it regardless of the activity of the other processes. Of all the problems in interprocess synchronization, the mutual exclusion problem is the one studied most extensively. This is a deceptive problem, and at first glance it seems very simple to solve.

Key Results

Numerous solutions for the problem have been proposed since it was first introduced by Edsger W. Dijkstra in 1965 [6]. Because of its importance and as a result of new hardware and software developments, new solutions to the problem are still being designed. Before the results are discussed, few models for interprocess communication are mentioned.

Atomic Operations

Most concurrent solutions to the problem assumes an architecture in which n processes communicate asynchronously via a shared objects. All architectures support *atomic registers*, which are shared objects that support atomic reads and writes operations. A weaker notion than an atomic register, called a *safe* register, is also considered in the literature. In a safe register, a read not concurrent with any writes must obtain the correct value, however, a read that is concurrent with some write, may return an arbitrary value. Most modern architectures support also some form of atomicity which is stronger than simple reads and writes. Common atomic operations have special names. Few examples are,

- *Test-and-set*: takes a shared registers r and a value *val*. The value *val* is assigned to r, and the old value of r is returned.
- *Swap*: takes a shared registers r and a local register ℓ, and atomically exchange their values.
- *Fetch-and-increment*: takes a register r. The value of r is incremented by 1, and the old value of r is returned.
- *Compare-and-swap*: takes a register r, and two values: *new* and *old*. If the current value of the register r is equal to *old*, then the value of r is set to *new* and the value *true* is returned; otherwise r is left unchanged and the value *false* is returned.

Modern operating systems (such as Unix and Windows) implement synchronization mechanisms, such as semaphores, that simplify the implementation of mutual exclusion locks and hence the design of concurrent applications. Also, modern programming languages (such as Modula and Java) implement the monitor concept which

is a program module that is used to ensure exclusive access to resources.

Algorithms and Lower Bounds

There are hundreds of beautiful algorithms for solving the problem some of which are also very efficient. Only few are mentioned below. First algorithms that use only atomic registers, or even safe registers, are discussed.

The Bakery Algorithm. The Bakery algorithm is one of the most known and elegant mutual exclusion algorithms using only safe registers [9]. The algorithm satisfies the FIFO requirement, however it uses unbounded size registers. A modified version, called the Black-White Bakery algorithm, satisfies FIFO and uses bounded number of bounded size atomic registers [14].

Lower bounds. A space lower bound for solving mutual exclusion using only atomic registers is that: any deadlock-free mutual exclusion algorithm for n processes must use at least n shared registers [5]. It was also shown in [5] that this bound is tight. A time lower bound for any mutual exclusion algorithm using atomic registers is that: there is no a priori bound on the number of steps taken by a process in its entry code until it enters its critical section (counting steps only when no other process is in its critical section or exit code) [2]. Many other interesting lower bounds exist for solving mutual exclusion.

A Fast Algorithm. A *fast* mutual exclusion algorithm, is an algorithm in which in the absence of contention only a constant number of shared memory accesses to the shared registers are needed in order to enter and exit a critical section. In [10], a fast algorithm using atomic registers is described, however, in the presence of contention, the winning process may have to check the status of all other n processes before it is allowed to enter its critical section. A natural question to ask is whether this algorithm can be improved for the case where there is contention.

Adaptive Algorithms. Since the other contending processes are waiting for the winner, it is particularly important to speed their entry to the critical section, by the design of an *adaptive* mutual exclusion algorithm in which the time complexity is independent of the total number of processes and is governed only by the current degree of contention. Several (rather complex) adaptive algorithms using atomic registers are known [1,3,14]. (Notice that, the time lower bound mention earlier implies that no adaptive algorithm using only atomic registers exists when time is measured by counting *all* steps.)

Local-spinning Algorithms. Many algorithms include busy-waiting loops. The idea is that in order to wait, a process *spins* on a flag register, until some other process terminates the spin with a single write operation. Unfortunately, under contention, such spinning may generate lots of traffic on the interconnection network between the process and the memory. An algorithm satisfies local spinning if the only type of spinning required is local spinning. Local Spinning is the situation where a process is spinning on locally-accessible registers. Shared registers may be locally-accessible as a result of either coherent caching or when using distributed shared memory where shared memory is physically distributed among the processors.

Three local-spinning algorithms are presented in [4,8,11]. These algorithms use strong atomic operations (i. e., fetch-and-increment, swap, compare-and-swap), and are also called *scalable* algorithms since they are both local-spinning and adaptive. Performance studies done, have shown that these algorithms scale very well as contention increases. Local spinning algorithms using only atomic registers are presented in [1,3,14].

Only few representative results have been mentioned. There are dozens of other very interesting algorithms and lower bounds. All the results discussed above, and many more, are described details in [15]. There are also many results for solving mutual exclusion in distributed message passing systems [13].

Applications

Synchronization is a fundamental challenge in computer science. It is fast becoming a major performance and design issue for concurrent programming on modern architectures, and for the design of distributed and concurrent systems.

Concurrent access to resources shared among several processes must be synchronized in order to avoid interference between conflicting operations. Mutual exclusion *locks* (i. e., algorithms) are the de facto mechanism for concurrency control on concurrent applications: a process accesses the resource only inside a critical section code, within which the process is guaranteed exclusive access. The popularity of this approach is largely due the apparently simple programming model of such locks and the availability of implementations which are efficient and scalable. Essentially all concurrent programs (including operating systems) use various types of mutual exclusion locks for synchronization.

When using locks to protect access to a resource which is a large data structure (or a database), the *granularity* of synchronization is important. Using a single lock to protect the whole data structure, allowing only one process at a time to access it, is an example of *coarse-grained* synchronization. In contrast, *fine-grained* synchronization enables

to lock "small pieces" of a data structure, allowing several processes with non-interfering operations to access it concurrently. Coarse-grained synchronization is easier to program but is less efficient and is not fault-tolerant compared to fine-grained synchronization. Using locks may degrade performance as it enforces processes to wait for a lock to be released. In few cases of simple data structures, such as queues, stacks and counters, locking may be avoided by using lock-free data structures.

Cross References

- ► Registers
- ► Self-Stabilization

Recommended Reading

In 1968, Edsger Wybe Dijkstra has published his famous paper "Co-operating sequential processes" [7], that originated the field of concurrent programming. The mutual exclusion problem was first stated and solved by Dijkstra in [6], where the first solution for two processes, due to Dekker, and the first solution for n processes, due to Dijkstra, have appeared. In [12], a collection of some early algorithms for mutual exclusion are described. In [15], dozens of algorithms for solving the mutual exclusion problems and wide variety of other synchronization problems are presented, and their performance is analyzed according to precise complexity measures.

1. Afek, Y., Stupp, G., Touitou, D.: Long lived adaptive splitter and applications. Distrib. Comput. **30**, 67–86 (2002)
2. Alur, R., Taubenfeld, G.: Results about fast mutual exclusion. In: Proceedings of the 13th IEEE Real-Time Systems Symposium, December 1992, pp. 12–21
3. Anderson, J.H., Kim, Y.-J.: Adaptive mutual exclusion with local spinning. In: Proceedings of the 14th international symposium on distributed computing. Lect. Notes Comput. Sci. **1914**, 29–43, (2000)
4. Anderson, T.E.: The performance of spin lock alternatives for shared-memory multiprocessor. IEEE Trans. Parallel Distrib. Syst. **1**(1), 6–16 (1990)
5. Burns, J.N., Lynch, N.A.: Bounds on shared-memory for mutual exclusion. Inform. Comput. **107**(2), 171–184 (1993)
6. Dijkstra, E.W.: Solution of a problem in concurrent programming control. Commun. ACM **8**(9), 569 (1965)
7. Dijkstra, E.W.: Co-operating sequential processes. In: Genuys, F. (ed.) Programming Languages, pp. 43–112. Academic Press, New York (1968). Reprinted from: Technical Report EWD-123, Technological University, Eindhoven (1965)
8. Graunke, G., Thakkar, S.: Synchronization algorithms for shared-memory multiprocessors. IEEE Comput. **28**(6), 69–69 (1990)
9. Lamport, L.: A new solution of Dijkstra's concurrent programming problem. Commun. ACM **17**(8), 453–455 (1974)
10. Lamport, L.: A fast mutual exclusion algorithm. ACM Trans. Comput. Syst. **5**(1), 1–11 (1987)
11. Mellor-Crummey, J.M., Scott, M.L.: Algorithms for scalable synchronization on shared-memory multiprocessors. ACM Trans. Comput. Syst. **9**(1), 21–65 (1991)
12. Raynal, M.: Algorithms for mutual exclusion. MIT Press, Cambridge (1986). Translation of: Algorithmique du parallélisme, (1984)
13. Singhal, M.: A taxonomy of distributed mutual exclusion. J. Parallel Distrib. Comput. **18**(1), 94–101 (1993)
14. Taubenfeld, G.: The black-white bakery algorithm. In: 18th international symposium on distributed computing, October (2004). LNCS, vol. 3274, pp. 56–70. Springer, Berlin (2004)
15. Taubenfeld, G.: Synchronization algorithms and concurrent programming. Pearson Education – Prentice-Hall, Upper Saddle River (2006) ISBN: 0131972596

Connected Dominating Set

2003; Cheng, Huang, Li, Wu, Du

Xiuzhen Cheng[1], Feng Wang[2], Ding-Zhu Du[3]

[1] Department of Computer Science, The George Washington University, Washington, D.C., USA

[2] Mathematical Science and Applied Computing, Arizona State University at the West Capmus, Phoenix, AZ, USA

[3] Department of Computer Science, University of Dallas at Texas, Richardson, TX, USA

Keywords and Synonyms

Techniques for partition

Problem Definition

Consider a graph $G = (V, E)$. A subset C of V is called a *dominating set* if every vertex is either in C or adjacent to a vertex in C. If, furthermore, the subgraph induced by C is connected, then C is called a *connected dominating set*. A connected dominating set with a minimum cardinality is called a *minimum connected dominating set (MCDS)*. Computing a MCDS is an NP-hard problem and there is no polynomial-time approximation with performance ratio $\rho H(\Delta)$ for $\rho < 1$ unless $NP \subseteq DTIME(n^{O(\ln \ln n)})$ where H is the harmonic function and Δ is the maximum degree of the input graph [10].

A unit disk is a disk with radius one. A *unit disk graph (UDG)* is associated with a set of unit disks in the Euclidean plane. Each node is at the center of a unit disk. An edge exists between two nodes u and v if and only if $|uv| \leq 1$ where $|uv|$ is the Euclidean distance between u and v. This means that two nodes u and v are connected

with an edge if and only if u's disk covers v and v's disk covers u.

Computing an MCDS in a unit disk graph is still NP-hard. How hard is it to construct a good approximation for MCDS in unit disk graphs? Cheng et al. [5] answered this question by presenting a polynomial-time approximation scheme.

Historical Background

The connected dominating set problem has been studied in graph theory for many years [22]. However, recently it becomes a hot topic due to its application in wireless networks for virtual backbone construction [4]. Guha and Khuller [10] gave a two-stage greedy approximation for the minimum connected dominating set in general graphs and showed that its performance ratio is $3 + \ln \Delta$ where Δ is the maximum node degree in the graph. To design a one-step greedy approximation to reach a similar performance ratio, the difficulty is to find a submodular potential function. In [21], Ruan et al. successfully designed a one step greedy approximation that reaches a better performance ratio $c + \ln \Delta$ for any $c > 2$. Du et al. [6] showed that there exits a polynomial-time approximation with a performance ratio $a(1 + \ln \Delta)$ for any $a > 1$. The importance of those works is that the potential functions used in their greedy algorithm are non-submodular and they managed to complete its theoretical performance evaluation with fresh ideas.

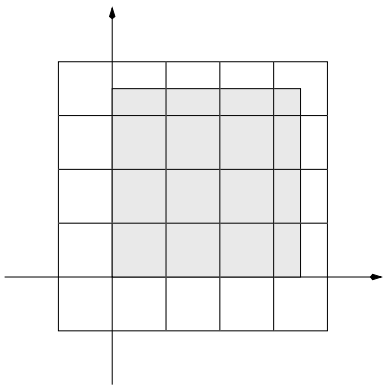
Guha and Khuller [10] also gave a negative result that there is no polynomial-time approximation with a performance ratio $\rho \ln \Delta$ for $\rho < 1$ unless $NP \subseteq DTIME(n^{O(\ln \ln n)})$. As indicated by [8], dominating sets cannot be approximated arbitrarily well, unless P almost equal to NP. These results move ones' attention from general graphs to unit disk graphs because the unit disk graph is the model for wireless sensor networks and in unit disk graphs, MCDS has a polynomial-time approximation with a constant performance ratio. While this constant ratio is getting improved step by step [1,2,19,24], Cheng et al. [5] closed this story by showing the existence of a polynomial-time approximation scheme (PTAS) for the MCDS in unit disk graphs. This means that theoretically, the performance ratio for polynomial-time approximation can be as small as $1 + \varepsilon$ for any positive number ε.

Dubhashi et al. [7] showed that once a dominating set is constructed, a connected dominating set can be easily computed in a distributed fashion. Most centralized results for dominating sets are available at [18]. In particular, a simple constant approximation for dominating sets in unit disk graphs was presented in [18]. Constant-factor approximation for minimum-weight (connected) dominating sets in UDGs was studied in [3]. A PTAS for the minimum dominating set problem in UDGs was proposed in [20]. Kuhn et al. [14] proved that a maximal independent set (MIS) (and hence also a dominating set) can be computed in asymptotically optimal time $O(\log n)$ in UDGs and a large class of bounded independence graphs. Luby [17] reported an elegant local $O(\log n)$ algorithm for MIS on general graphs. Jia et al. [11] proposed a fast $O(\log n)$ distributed approximation for dominating set in general graphs. The first constant-time distributed algorithm for dominating sets that achieves a non-trivial approximation ratio for general graphs was reported in [15]. The matching $\Omega(\log n)$ lower bound is considered to be a classic result in distributed computing [16]. For UDGs a PTAS is achievable in a distributed fashion [13]. The fastest deterministic distributed algorithm for dominating sets in UDGs was reported in [12], and the fastest randomized distributed algorithm for dominating sets in UDGs was presented in [9].

Key Results

The construction of PTAS for MCDS is based on the fact that there is a polynomial-time approximation with a constant performance ratio. Actually, this fact is quite easy to see. First, note that a unit disk contains at most five independent vertices [2]. This implies that every maximal independent set has a size at most $1 + 4opt$ where opt is the size of an MCDS. Moreover, every maximal independent set is a dominating set and it is easy to construct a maximal independent set with a spanning tree of all edges with length two. All vertices in this spanning tree form a connected dominating set of a size at most $1 + 8opt$. By improving the upper bound for the size of a maximal independent set [25] and the way to interconnecting a maximal independent set [19], the constant ratio has been improved to 6.8 with a distributed implementation.

The basic techniques in this construction is nonadaptive partition and shifting. Its general picture is as follows: First, the square containing all vertices of the input unit-disk graph is divided into a grid of small cells. Each small cell is further divided into two areas, the central area and the boundary area. The central area consists of points h distance away from the cell boundary. The boundary area consists of points within distance $h + 1$ from the boundary. Therefore, two areas are overlapping. Then a minimum union of connected dominating sets is computed in each cell for connected components of the central area of the cell. The key lemma is to



Connected Dominating Set, Figure 1
Squares Q and $\bar{Q}$

prove that the union of all such minimum unions is no more than the minimum connected dominating set for the whole graph. For vertices not in central areas, just use the part of an 8-approximation lying in boundary areas to dominate them. This part together with the above union forms a connected dominating set for the whole input unit-disk graph. By shifting the grid around to get partitions at different coordinates, a partition having the boundary part with a very small upper bound can be obtained.

The following details the construction.

Given an input connected unit-disk graph $G = (V, E)$ residing in a square $Q = \{(x, y) \mid 0 \le x \le q, 0 \le y \le q\}$ where $q \le |V|$. To construct an approximation with a performance ratio $1 + \varepsilon$ for $\varepsilon > 0$, choose an integer $m = O((1/\varepsilon)\ln(1/\varepsilon))$. Let $p = \lfloor q/m \rfloor + 1$. Consider the square $\bar{Q} = \{(x, y) \mid -m \le x \le mp, -m \le y \le mp\}$. Partition $\bar{Q}$ into $(p + 1) \times (p + 1)$ grids so that each cell is an $m \times m$ square excluding the top and the right boundaries and hence no two cells are overlapping each other. This partition of $\bar{Q}$ is denoted by $P(0)$ (Fig. 1). In general, the partition $P(a)$ is obtained from $P(0)$ by shifting the bottom-left corner of $\bar{Q}$ from $(-m, -m)$ to $(-m + a, -m + a)$. Note that shifting from $P(0)$ to $P(a)$ for $0 \le a \le m$ keeps Q covered by the partition.

For each cell e (an $m \times m$ square), $C_e(d)$ denotes the set of points in e away from the boundary by distance at least d, e. g., $C_e(0)$ is the cell e itself. Denote $B_e(d) = C_e(0) - C_e(d)$. Fix a positive integer $h = 7 + 3\lfloor \log_2(4m^2/\pi) \rfloor$. Call $C_e(h)$ the *central area* of e and $B_e(h + 1)$ the *boundary area* of e. Hence the boundary area and the central area of each cell are overlapping with width one.

Central Area

Let $G_e(d)$ denote the part of input graph G lying in area $C_e(d)$. In particular, $G_e(h)$ is the part of graph G lying in the central area of e. $G_e(h)$ may consist of several connected components. Let K_e be a subset of vertices in $G_e(0)$ with a minimum cardinality such that for each connected component H of $G_e(h)$, K_e contains a connected component dominating H. In other words, K_e is a minimum union of connected dominating sets in $G(0)$ for the connected components of $G_e(h)$.

Now, denote by $K(a)$ the union of K_e for e over all cells in partition $P(a)$. $K(a)$ has two important properties:

Lemma 1 *$K(a)$ can be computed in time $n^{O(m^2)}$.*

Lemma 2 *$|K^a| \le opt$ for $0 \le a \le m - 1$.*

Lemma 1 is not hard to see. Note that in a square with edge length $\sqrt{2}/2$, all vertices induce a complete subgraph in which any vertex must dominate all other vertices. It follows that the minimum dominating set for the vertices of $G_e(0)$ has size at most $(\lceil \sqrt{2}m \rceil)^2$. Hence, the size of K_e is at most $3(\lceil \sqrt{2}m \rceil)^2$ because any dominating set in a connected graph has a spanning tree with an edge length at most three. Suppose cell $G_e(0)$ has n_e vertices. Then the number of candidates for K_e is at most

$$\sum_{k=0}^{3(\lceil \sqrt{2}m \rceil)^2} \binom{n_e}{k} = n_e^{O(m^2)}.$$

Hence, computing $K(a)$ can be done in time

$$\sum_e n_e^{O(m^2)} \le \left(\sum_e n_e \right)^{O(m^2)} = n^{O(m^2)}.$$

However, the proof of Lemma 2 is quite tedious. The reader who is interested in it may find it in [5].

Boundary Area

Let F be a connected dominating set of G satisfying $|F| \le 8opt + 1$. Denote by $F(a)$ the subset of F lying in the boundary area $B_a(h + 1)$. Since F is constructed in polynomial-time, only the size of $F(a)$ needs to be studied.

Lemma 3 *Suppose $h = 7 + 3\lfloor \log_2(4m^2/\pi) \rfloor$ and $\lfloor m/(h + 1) \rfloor \ge 32/\varepsilon$. Then there is at least half of $i = 0, 1, \ldots, \lfloor m/(h+1) \rfloor - 1$ such that $|F(i(h + 1))| \le \varepsilon \cdot opt$.*

Proof Let $F_H(a)$ ($F_V(a)$) denote the subset of vertices in $F(a)$ each with distance $< h + 1$ from the horizontal (vertical) boundary of some cell in $P(a)$. Then

$F(a) = F_H(a) \cup F_V(a)$. Moreover, all $F_H(i(h+1))$ for $i = 0, 1, ..., \lfloor m/(h+1) \rfloor - 1$ are disjoint. Hence,

$$\sum_{i=0}^{\lfloor m/(h+1) \rfloor - 1} |F_H(i(h+1))| \le |F| \le 8opt.$$

Similarly, all $F_V(i(h+1))$ for $i = 0, 1, ..., \lfloor m/(h+1) \rfloor - 1$ are disjoint and

$$\sum_{i=0}^{\lfloor m/(h+1) \rfloor - 1} |F_V(i(h+1))| \le |F| \le 8opt\,.$$

Thus

$$\begin{aligned} &\sum_{i=0}^{\lfloor m/(h+1) \rfloor - 1} |F(i(h+1))| \\ \le &\sum_{i=0}^{\lfloor m/(h+1) \rfloor - 1} (|F_H(i(h+1))| + |F_V(i(h+1))|) \\ \le &\ 16opt\,. \end{aligned}$$

That is,

$$\frac{1}{\lfloor m/(h+1) \rfloor} \sum_{i=0}^{\lfloor m/(h+1) \rfloor - 1} |F(i(h+1))| \le (\varepsilon/2)opt.$$

This means that there are at least half of $F(i(h+1))$ for $i = 0, 1, \lfloor m/(h+1) \rfloor - 1$ satisfying

$$|F(i(h+1))| \le \varepsilon \cdot opt\,. \qquad \square$$

Putting Together

Now put $K(a)$ and $F(a)$. By Lemmas 2 and 3, there exists $a \in \{0, h+1, ..., (\lfloor m/(h+1) \rfloor - 1)(h+1)\}$ such that

$$|K(a) \cup F(a)| \le (1+\varepsilon)opt.$$

Lemma 4 *For $0 \le a \le m-1$, $K(a) \cup F(a)$ is a connected dominating for input connected graph G.*

Proof $K(a) \cup F(a)$ is clearly a dominating set for input graph G. Its connectivity can be shown as follows. Note that the central area and the boundary area are overlapping with an area of width one. Thus, for any connected component H of the subgraph $G_e(h)$, $F(a)$ has a vertex in H. Hence, $F(a)$ must connect to any connected dominating set for H, especially, the one D_H in $K(a)$. This means that D_H has making up the connections of F lost from cutting a part in H. Therefore, the connectivity of $K(a) \cup F(a)$ follows from the connectivity of F. $\square$

By summarizing the above results, the following result is obtained:

Theorem 1 *There is a $(1+\varepsilon)$-approximation for MCDS in connected unit-disk graphs, running in time $n^{O((1/\varepsilon)\log(1/\varepsilon)^2)}$.*

Applications

An important application of connected dominating sets is to construct virtual backbones for wireless networks, especially, wireless sensor networks [4]. The topology of a wireless sensor network is often a unit disk graph.

Open Problems

In general, the topology of a wireless network is a disk graph, that is, each vertex is associated with a disk. Different disks may have different sizes. There is an edge from vertex u to vertex v if and only if the disk at u covers v. A virtual backbone in disk graphs is a subset of vertices, which induces a strongly connected subgraph, such that every vertex not in the subset has an in-edge coming from a vertex in the subset and also has an out-edge going into a vertex in the subset. Such a virtual backbone can be considered as a *connected dominating set* in disk graph. Is there a polynomial-time approximation with a constant performance ratio? It is open right now. Thai et al. [23] has made some effort towards this direction.

Cross References

- ▶ Dominating Set
- ▶ Exact Algorithms for Dominating Set
- ▶ Greedy Set-Cover Algorithms
- ▶ Max Leaf Spanning Tree

Recommended Reading

1. Alzoubi, K.M., Wan, P.-J., Frieder, O.: Message-optimal connected dominating sets in mobile ad hoc networks. In: ACM MOBIHOC, Lausanne, Switzerland, 09–11 June 2002
2. Alzoubi, K.M., P.-J.Wan, Frieder, O.: New Distributed Algorithm for Connected Dominating Set in Wireless Ad Hoc Networks. In: HICSS35, Hawaii, January 2002
3. Ambuhl, C., Erlebach, T., Mihalak, M., Nunkesser, M.: Constant-Factor Approximation for Minimum-Weight (Connected) Dominating Sets in Unit Disk Graphs. In: LNCS, vol. 4110, pp 3–14. Springer, Berlin (2006)
4. Blum, J., Ding, M., Thaeler, A., Cheng, X.: Applications of Connected Dominating Sets in Wireless Networks. In: Du, D.-Z., Pardalos, P. (eds.) Handbook of Combinatorial Optimization, pp. 329–369. Kluwer Academic (2004)
5. Cheng, X., Huang, X., Li, D., Wu, W., Du, D.-Z.: A polynomial-time approximation scheme for minimum connected dominating set in ad hoc wireless networks. Networks **42**, 202–208 (2003)

6. Du, D.-Z., Graham, R.L., Pardalos, P.M., Wan, P.-J., Wu, W., Zhao, W.: Analysis of greedy approximations with nonsubmodular potential functions. In: Proceedings of the 19th annual ACM-SIAM Symposium on Discrete Algorithms (SODA) pp. 167–175. January 2008
7. Dubhashi, D., Mei, A., Panconesi, A., Radhakrishnan, J., Srinivasan, A.: Fast Distributed Algorithms for (Weakly) Connected Dominating Sets and Linear-Size Skeletons. In: SODA, 2003, pp. 717–724
8. Feige, U.: A Threshold of $\ln n$ for Approximating Set Cover. J. ACM **45**(4) 634–652 (1998)
9. Gfeller, B., Vicari, E.: A Randomized Distributed Algorithm for the Maximal Independent Set Problem in Growth-Bounded Graphs. In: PODC 2007
10. Guha, S., Khuller, S.: Approximation algorithms for connected dominating sets. Algorithmica **20**, 374–387 (1998)
11. Jia, L., Rajaraman, R., Suel, R.: An Efficient Distributed Algorithm for Constructing Small Dominating Sets. In: PODC, Newport, Rhode Island, USA, August 2001
12. Kuhn, F., Moscibroda, T., Nieberg, T., Wattenhofer, R.: Fast Deterministic Distributed Maximal Independent Set Computation on Growth-Bounded Graphs. In: DISC, Cracow, Poland, September 2005
13. Kuhn, F., Moscibroda, T., Nieberg, T., Wattenhofer, R.: Local Approximation Schemes for Ad Hoc and Sensor Networks. In: DIALM-POMC, Cologne, Germany, September 2005
14. Kuhn, F., Moscibroda, T., Wattenhofer, R.: On the Locality of Bounded Growth. In: PODC, Las Vegas, Nevada, USA, July 2005
15. Kuhn, F., Wattenhofer, R.: Constant-Time Distributed Dominating Set Approximation. In: PODC, Boston, Massachusetts, USA, July 2003
16. Linial, N.: Locality in distributed graph algorithms. SIAM J. Comput. **21**(1), 193–201 (1992)
17. Luby, M.: A Simple Parallel Algorithm for the Maximal Independent Set Problem. SIAM J. Comput. **15**, 1036–1053 (1986)
18. Marathe, M.V., Breu, H., Hunt III, H.B., Ravi, S.S., Rosenkrantz, D.J.: Simple Heuristics for Unit Disk Graphs. Networks **25**, 59–68 (1995)
19. Min, M., Du, H., Jia, X., Huang, X., Huang, C.-H., Wu, W.: Improving construction for connected dominating set with Steiner tree in wireless sensor networks. J. Glob. Optim. **35**, 111–119 (2006)
20. Nieberg, T., Hurink, J.L.: A PTAS for the Minimum Dominating Set Problem in Unit Disk Graphs. LNCS, vol. 3879, pp. 296–306. Springer, Berlin (2006)
21. Ruan, L., Du, H., Jia, X., Wu, W., Li, Y., Ko, K.-I.: A greedy approximation for minimum connected dominating set. Theor. Comput. Sci. **329**, 325–330 (2004)
22. Sampathkumar, E., Walikar, H.B.: The Connected Domination Number of a Graph. J. Math. Phys. Sci. **13**, 607–613 (1979)
23. Thai, M.T., Wang F., Liu, D., Zhu, S., Du, D.-Z.: Connected Dominating Sets in Wireless Networks with Different Transmission Range. IEEE Trans. Mob. Comput. **6**(7), 721–730 (2007)
24. Wan, P.-J., Alzoubi, K.M., Frieder, O.: Distributed Construction of Connected Dominating Set in Wireless Ad Hoc Networks. In: IEEE INFOCOM 2002
25. Wu, W., Du, H., Jia, X., Li, Y., Huang, C.-H.: Minimum Connected Dominating Sets and Maximal Independent Sets in Unit Disk Graphs. Theor. Comput. Sci. **352**, 1–7 (2006)

Connectivity and Fault-Tolerance in Random Regular Graphs

2000; Nikoletseas, Palem, Spirakis, Yung

SOTIRIS NIKOLETSEAS
Department of Computer Engineering and Informatics, Computer Technology Institute, University of Patras and CTI, Patras, Greece

Keywords and Synonyms

Robustness

Problem Definition

A new model of random graphs was introduced in [7], that of random regular graphs with edge faults (denoted hereafter by $G^r_{n,p}$), obtained by selecting the edges of a random member of the set of all regular graphs of degree r independently and with probability p. Such graphs can represent a communication network in which the links fail independently and with probability $f = 1 - p$. A formal definition of the probability space $G^r_{n,p}$ follows.

Definition 1 (the $G^r_{n,p}$ probability space) Let G^r_n be the probability space of all random regular graphs with n vertices where the degree of each vertex is r. The probability space $G^r_{n,p}$ of random regular graphs with edge faults is constructed by the following two subsequent random experiments: first, a random regular graph is chosen from the space G^r_n and, second, each edge is randomly and independently deleted from this graph with probability $f = 1 - p$.

Important connectivity properties of $G^r_{n,p}$ are investigated in this entry by estimating the ranges of r, f for which, with high probability, $G^r_{n,p}$ graphs a) are highly connected b) become disconnected and c) admit a giant (i. e. of $\Theta(n)$ size) connected component of small diameter.

Notation The terms "almost certainly" (a.c.) and "with high probability" (w.h.p.) will be frequently used with their standard meaning for random graph properties. A property defined in a random graph holds almost certainly when its probability tends to 1 as the independent variable (usually the number of vertices in the graph) tends to infinity. "With high probability" means that the probability of a property of the random graph (or the success probability of a randomized algorithm) is at least $1 - n^{-\alpha}$, where $\alpha > 0$ is a constant and n is the number of vertices in the graph.

The interested reader can further study [1] for an excellent exposition of the Probabilistic Method and its applications, [2] for a classic book on random graphs, as well

as [6], an excellent book on the design and analysis of randomized algorithms.

Key Results

Summary This entry studies several important connectivity properties of random regular graphs with edge faults. In order to deal with the $G^r_{n,p}$ model, [7] first extends the notion of configurations and the translation lemma between configurations and random regular graphs provided by B. Bollobás [2,3], by introducing the concept of *random configurations* to account for edge faults, and by also providing an *extended translation lemma* between random configurations and random regular graphs with edge faults.

For this new model of random regular graphs with edge faults [7] shows that:

1. For all failure probabilities $f = 1 - p \leq n^{-\epsilon}$ ($\epsilon \geq \frac{3}{2r}$ fixed) and any $r \geq 3$ the biggest part of $G^r_{n,p}$ (i. e. the whole graph except of $O(1)$ vertices) remains connected and this connected part can not be separated, almost certainly, unless more than r vertices are removed. Note interestingly that the situation for this range of f and r is very similar, despite the faults, to the properties of G^r_n which is r-connected for $r \geq 3$.
2. $G^r_{n,p}$ is *disconnected* a.c. for constant f and any $r = o(\log n)$, but is *highly connected*, almost certainly, when $r \geq \alpha \log n$, where $\alpha > 0$ an appropriate constant.
3. Even when $G^r_{n,p}$ becomes disconnected, it still has *a giant component of small diameter*, even when $r = O(1)$. An $O(n \log n)$-time algorithm to construct a giant component is provided.

Configurations and Translation Lemmata

Note that it is not as easy (from the technical point of view) as in the $G_{n,p}$ case to argue about random regular graphs, because of the stochastic dependencies on the existence of the edges due to regularity. The following notion of *configurations* was introduced by B. Bollobás [2,3] to translate statements for random regular graphs to statements for the corresponding configurations which avoid the edge dependencies due to regularity and thus are much easier to deal with:

Definition 2 (Bollobás, [3]) Let $w = \cup_{j=1}^{n} w_j$ be a fixed set of $2m = \sum_{j=1}^{n} d_j$ labeled vertices where $|w_j| = d_j$. A configuration F is a partition of w into m pairs of vertices, called edges of F.

Given a configuration F, let $\theta(F)$ be the (multi)graph with vertex set V in which (i,j) is an edge if and only if F has a pair (edge) with one element in w_i and the other in w_j. Note that every regular graph $G \in G^r_n$ is of the form $\theta(F)$ for exactly $(r!)^n$ configurations. However not every configuration F with $d_j = r$ for all j corresponds to a $G \in G^r_n$ since F may have an edge entirely in some w_j or parallel edges joining w_i and w_j.

Let φ be the set of all configurations F and let G^r_n be the set of all regular graphs. Given a property (set) $Q \subseteq G^r_n$ let $Q^* \subseteq \phi$ such that $Q^* \cap \theta^{-1}(G^r_n) = \theta^{-1}(Q)$. By estimating the probability of possible cycles of length one (self-loops) and two (loops) among pairs w_i, w_j in $\theta(F)$, The following important lemma follows:

Lemma 1 (Bollobás, [2]) *If $r \geq 2$ is fixed and property Q^* holds for a.e. configuration, then property Q holds for a.e. $r-$regular graph.*

The main importance of the above lemma is that when studying random regular graphs, instead of considering the set of all random regular graphs, one can study the (much more easier to deal with) set of configurations.

In order to deal with edge failures, [7] introduces here the following extension of the notion of configurations:

Definition 3 (random configurations) Let $w = \cup_{j=1}^{n} w_j$ be a fixed set of $2m = \sum_{j=1}^{n} d_j$ labeled "vertices" where $|w_j| = d_j$. Let F be any configuration of the set φ. For each edge of F, remove it with probability $1 - p$, independently. Let $\hat{\phi}$ be the new set of objects and $\hat{F}$ the outcome of the experiment. $\hat{F}$ is called a *random configuration*.

By introducing probability p in every edge, an extension of the proof of Lemma 1 leads (since in both $\bar{Q}$ and $\hat{Q}$ each edge has the same probability and independence to be deleted, thus the modified spaces follow the properties of Q and Q^*) to the following extension to random configurations.

Lemma 2 (extended translation lemma) *Let $r \geq 2$ fixed and $\bar{Q}$ be a property for $G^r_{n,p}$ graphs. If $\hat{Q}$ holds for a.e. random configuration, then the corresponding property $\bar{Q}$ holds for a.e. graph in $G^r_{n,p}$.*

Multiconnectivity Properties of $G^r_{n,p}$

The case of constant link failure probability f is studied, which represents a worst case for connectivity preservation. Still, [7] shows that logarithmic degrees suffice to guarantee that $G^r_{n,p}$ remains w.h.p. highly connected, despite these constant edge failures. More specifically:

Theorem 3 *Let G be an instance of $G^r_{n,p}$ where $p = \Theta(1)$ and $r \geq \alpha \log n$, where $\alpha > 0$ an appropriate constant.*

Then G is almost certainly k-connected, where

$$k = O\left(\frac{\log n}{\log\log n}\right).$$

The proof of the above Theorem uses Chernoff bounds to estimate the vertex degrees in $G^r_{n,p}$, and "similarity" of $G^r_{n,p}$ and $G_{n,p'}$ (whose properties are known) for a suitably chosen p'.

Now the (more practical) case in which $f = 1 - p = o(1)$ is considered and it is proved that the desired connectivity properties of random regular graphs are almost preserved despite the link failures. More specifically:

Theorem 4 *Let $r \geq 3$ and $f = 1 - p = O(n^{-\epsilon})$ for $\epsilon \geq \frac{3}{2r}$. Then the biggest part of $G^r_{n,p}$ (i. e. the whole graph except of O(1) vertices) remains connected and this connected part (excluding the vertices that were originally neighbors of the O(1)-sized disconnected set) can not be separated unless more than r vertices are removed, with probability tending to 1 as n tends to $+\infty$.*

The proof is carefully extending, in the case of faults, a known technique for random regular graphs about not admitting small separators.

$G^r_{n,p}$ Becomes Disconnected

Next remark that a constant link failure probability dramatically alters the connectivity structure of the regular graph in the case of low degrees. In particular, by using the notion of random configurations, [7] proves the following theorem:

Theorem 5 *When $2 \leq r \leq \frac{\sqrt{\log n}}{2}$ and $p = \Theta(1)$ then $G^r_{n,p}$ has at least one isolated node with probability at least $1 - n^{-k}$, $k \geq 2$.*

The regime for disconnection is in fact larger, since [7] shows that $G^r_{n,p}$ is a.c. disconnected even for any $r = o(\log n)$ and constant f. The proof of this last claim is complicated by the fact that due to the range for r one has to avoid using the extended translation lemma.

Existence of a Giant Component in $G^r_{n,p}$

Since $G^r_{n,p}$ is a.c. disconnected for $r = o(\log n)$ and $1 - p = f = \Theta(1)$, it would be interesting to know whether at least a large part of the network represented by $G^r_{n,p}$ is still connected, i. e. whether the biggest connected component of $G^r_{n,p}$ is large. In particular, [7] shows that:

Theorem 6 *When $f < 1 - \frac{32}{r}$ then $G^r_{n,p}$ admits a giant (i. e. $\Theta(n)$-sized) connected component for any $r \geq 64$ with probability at least $1 - O(\log^2 n)/(n^{\alpha/3})$, where $\alpha > 0$ a constant that can be selected.*

In fact, the proof of the existence of the component includes first proving the existence (w.h.p.) of a sufficiently long (of logarithmic size) path as a basis for a BFS process starting from the vertices of that path that creates the component. The proof is quite complex: occupancy arguments are used (bins correspond to the vertices of the graphs while balls correspond to its edges); however, the random variables involved are not independent, and in order to use Chernoff-Hoeffding bounds for concentration one must prove that these random variables, although not independent, are negatively associated. Furthermore, the evaluation of the success of the BFS process uses a careful, detailed average case analysis.

The path construction and the BFS process can be viewed as an algorithm that (in case of no failures) actually reveals a giant connected component. This algorithm is very efficient, as shown by the following result:

Theorem 7 *A giant component of $G^r_{n,p}$ can be constructed in $O(n \log n)$ time, with probability at least $1 - O(\log^2 n)/(n^{\alpha/3})$, where $\alpha > 0$ a constant that can be selected.*

Applications

In recent years the development and use of distributed systems and communication networks has increased dramatically. In addition, state-of-the-art multiprocessor architectures compute over structured, regular interconnection networks. In such environments, several applications may share the same network while executing concurrently. This may lead to unavailability of certain network resources (e. g. links) for certain applications. Similarly, faults may cause unavailability of links or nodes. The aspect of *reliable distributed computing* (which means computing with the available resources and resisting faults) adds value to applications developed in such environments.

When computing in the presence of faults, one cannot assume that the actual structure of the computing environment is known. Faults may happen even in execution time. In addition, what is a "faulty" or "unavailable" link for one application may in fact be the de-allocation of that link because it is assigned (e. g. by the network operation system) to another application. The problem of analyzing allocated computation or communication in a network over a *randomly assigned subnetwork* and *in the presence of faults* has a nature different from fault analysis of special, well-structured networks (e. g. hypercube), which does not deal with network aspects. The work presented in this entry

addresses this interesting issue, i. e. analyzing the average case taken over a set of possible topologies and focuses on multiconnectivity and existence of giant component properties, required for reliable distributed computing in such randomly allocated unreliable environments.

The following important application of this work should be noted: multitasking in distributed memory multiprocessors is usually performed by assigning an arbitrary subnetwork (of the interconnection network) to each task (called the *computation graph*). Each parallel program may then be expressed as communicating processors over the computation graph. Note that a multiconnectivity value k of the computation graph means also that the execution of the application can tolerate up to $k-1$ *on-line additional faults*.

Open Problems

The ideas presented in [7] inspired already further interesting research. Andreas Goerdt [4] continued the work presented in a preliminary version [8] of [7] and showed the following results: if the degree r is fixed then $p = \frac{1}{r-1}$ is a threshold probability for the existence of a linear sized component in the faulty version of almost all random regular graphs. In fact, he further shows that if each edge of an *arbitrary* graph G with maximum degree bounded above by r is present with probability $p = \frac{\lambda}{r-1}$, when $\lambda < 1$, then the faulty version of G has only components whose size is at most logarithmic in the number of nodes, with high probability. His result implies some kind of optimality of random regular graphs with edge faults. Furthermore, [5,10] investigates important expansion properties of random regular graphs with edge faults, as well as [9] does in the case of fat-trees, a common type of interconnection networks. It would be also interesting to further pursue this line of research, by also investigating other combinatorial properties (and also provide efficient algorithms) for random regular graphs with edge faults.

Cross References

- ▶ Hamilton Cycles in Random Intersection Graphs
- ▶ Independent Sets in Random Intersection Graphs
- ▶ Minimum k-Connected Geometric Networks

Recommended Reading

1. Alon, N., Spencer, J.: The Probabilistic Method. Wiley (1992)
2. Bollobás, B.: Random Graphs. Academic Press (1985)
3. Bollobás, B.: A probabilistic proof of an asymptotic formula for the number of labeled regular graphs. Eur. J. Comb. **1**, 311–316 (1980)
4. Goerdt, A.: The giant component threshold for random regular graphs with edge faults. In: Proceedings of Mathematical Foundations of Computer Science '97 (MFCS'97), pp. 279–288. (1997)
5. Goerdt, A.: Random regular graphs with edge faults: Expansion through cores. Theor. Comput. Sci. **264**(1), 91–125 (2001)
6. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)
7. Nikoletseas, S., Palem, K., Spirakis, P., Yung, M.: Connectivity Properties in Random Regular Graphs with Edge Faults. In: Special Issue on Randomized Computing of the International Journal of Foundations of Computer Science (IJFCS), vol. 11 no. 2, pp. 247–262, World Scientific Publishing Company (2000)
8. Nikoletseas, S., Palem, K., Spirakis, P., Yung, M.: Short Vertex Disjoint Paths and Multiconnectivity in Random Graphs: Reliable Network Computing. In: Proc. 21st International Colloquium on Automata, Languages and Programming (ICALP), pp. 508–515. Jerusalem (1994)
9. Nikoletseas, S., Pantziou, G., Psycharis, P., Spirakis, P.: On the reliability of fat-trees. In: Proc. 3rd International European Conference on Parallel Processing (Euro-Par), pp. 208–217, Passau, Germany (1997)
10. Nikoletseas, S., Spirakis, P.: Expander Properties in Random Regular Graphs with Edge Faults. In: Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS), pp.421–432, München (1995)

Consensus with Partial Synchrony

1988; Dwork, Lynch, Stockmeyer

Bernadette Charron-Bost[1], André Schiper[2]

[1] Laboratory for Informatics, The Polytechnic School, Palaiseau, France

[2] EPFL, Lausanne, Switzerland

Keywords and Synonyms

Agreement problem

Problem Definition

Reaching agreement is one of the central issues in fault tolerant distributed computing. One version of this problem, called *Consensus*, is defined over a fixed set $\Pi = \{p_1, \ldots, p_n\}$ of n processes that communicate by exchanging messages along channels. Messages are correctly transmitted (no duplication, no corruption), but some of them may be lost. Processes may fail by prematurely stopping (crash), may omit to send or receive some messages (omission), or may compute erroneous values (Byzantine faults). Such processes are said to be *faulty*. Every process $p \in \Pi$ has an initial value v_p and

non-faulty processes must decide irrevocably on a common value v. Moreover, if the initial values are all equal to the same value v, then the common decision value is v. The properties that define Consensus can be split into safety properties (processes decide on the same value; the decision value must be consistent with initial values) and a liveness property (processes must eventually decide).

Various Consensus algorithms have been described [6,12] to cope with any type of process failures if there is a known[1] bound on the transmission delay of messages (*communication is synchronous*) and a known bound on process relative speeds (*processes are synchronous*). In completely asynchronous systems, where there exists no bound on transmission delays and no bound on process relative speeds, Fischer, Lynch, and Paterson [8] have proved that there is no Consensus algorithm resilient to even one crash failure. The paper by Dwork, Lynch, and Stockmeyer [7] introduces the concept of *partial synchrony*, in the sense it lies between the completely synchronous and completely asynchronous cases, and shows that partial synchrony makes it possible to solve Consensus in the presence of process failures, whatever the type of failure is.

For this purpose, the paper examines the quite realistic case of asynchronous systems that behave synchronously during some "good" periods of time. Consensus algorithms designed for synchronous systems do not work in such systems since they may violate the safety properties of Consensus during a bad period, that is when the system behaves asynchronously. This leads to the following question: is it possible to design a Consensus algorithm that never violates safety conditions in an asynchronous system, while ensuring the liveness condition when some additional conditions are met?

Key Results

The paper has been the first to provide a positive and comprehensive answer to the above question. More precisely, the paper (1) defines various types of partial synchrony and introduces a new round based computational model for partially synchronous systems, (2) gives various Consensus algorithms according to the severity of failures (crash, omission, Byzantine faults with or without authentication), and (3) shows how to implement the round based computational model in each type of partial synchrony.

[1]Intuitively, "known bound" means that the bound can be "built into" the algorithm. A formal definition is given in the next section.

Partial Synchrony

Partial synchrony applies both to communications and to processes. Two definitions for *partially synchronous communications* are given: (1) for each run, there exists an upper bound Δ on communication delays, but Δ is *unknown* in the sense it depends on the run; (2) there exists an upper bound Δ on communication delays that is common for all runs (Δ is *known*), but holds only after some time T, called the *Global Stabilization Time* (*GST*) that may depend on the run (*GST* is *unknown*). Similarly, *partially synchronous processes* are defined by replacing "transmission delay of messages" by "relative process speeds" in (1) and (2) above. That is, the upper bound on relative process speed Φ is unknown, or Φ is known but holds only after some unknown time.

Basic Round Model

The paper considers a round based model: computation is divided into *rounds* of message exchange. Each round consists of a *send step*, a *receive step*, and then a *computation step*. In a send step, each process sends messages to any subset of processes. In a receive step, some subset of the messages sent to the process during the send step at the *same* round is received. In a computation step, each process executes a state transition based on its current state and the set of messages just received.

Some of the messages that are sent may not be received, i.e, some can be lost. However, the *basic round model* assumes that there is some round GSR, such that all messages sent from non faulty processes to non faulty processes at round GSR or afterward are received.

Consensus Algorithm for Benign Faults (requires $f < n/2$)

In the paper, the algorithm is only described informally (textual form). A formal expression is given by Algorithm 1: the code of each process is given round by round, and each round is specified by the send and the computation steps (the receive step is implicit). The constant f denotes the maximum number of processes that may be faulty (crash or omission). The algorithm requires $f < n/2$.

Rounds are grouped into phases, where each phase consists in four consecutive rounds. The algorithm includes the rotating coordinator strategy: each phase k is led by a unique coordinator—denoted by $coord_k$—defined as process p_i for phase $k = i(mod\ n)$. Each process p maintains a set $Proper_p$ of values that p has heard of (*proper* values), initialized to $\{v_p\}$ where v_p is p's ini-

```
1:  Initialization:
2:      Acceptable_p := {v_p}                                   {v_p is the initial value of p }
3:      Proper_p := {v_p}                                       {All the lines for maintaining Proper_p are trivial to write, and so are omitted}
4:      vote_p := ⊥
5:      Lock_p := ∅

6:  Round r = 4k − 3:
7:      Send:
8:          send ⟨Acceptable_p⟩ to coord_k

9:      Compute:
10:         if p = coord_k and p receives at least ≥ n − f messages containing a common value then
11:             vote_p := select one of these common acceptable values

12: Round r = 4k − 2:
13:     Send:
14:         if p = coord_k and vote_p ≠ ⊥ then
15:             send ⟨vote_p⟩ to all processes

16:     Compute:
17:         if received ⟨v⟩ from coord_k then
18:             Lock_p := Lock_p \ {v, −}; Lock_p := Lock_p ∪ {(v, k)};

19: Round r = 4k − 1:
20:     Send:
21:         if ∃v s.t. (v , k) ∈ Lock_p then
22:             send ⟨ack⟩ to coord_k

23:     Compute:
24:         if p = coord_k then
25:             if received at least ≥ f + 1 ack messages then
26:                 DECIDE(vote_p );
27:             vote_p := ⊥

28: Round r = 4k:
29:     Send:
30:         send ⟨Lock_p⟩ to all processes

31:     Compute:
32:         for all (v, θ) ∈ Lock_p do
33:             if received (w, θ̄) s.t. w ≠ v and θ̄ ≥ θ then          {release lock on v}
34:                 Lock_p := Lock_p ∪ {(w, θ̄)} \ {(v, θ)};
35:         if |Lock_p| = 1 then
36:             Acceptable_p := v where (v, −) ∈ Lock_p
37:         else
38:             if Lock_p = ∅ then Acceptable_p := Proper_p else Acceptable_p := ∅
```

Consensus with Partial Synchrony, Algorithm 1
Consensus algorithm in the basic round model for benign faults ($f < n/2$)

tial value. Process p attaches $Proper_p$ to each message it sends.

Process p may *lock* value v when p thinks that some process might decide v. Thus value v is an *acceptable* value to p if (1) v is a proper value to p, and (2) p does not have a lock on any value except possibly v (lines 35 to 38).

At the first round of phase k (round $4k-3$), each process sends the list of its acceptable values to $coord_k$. If $coord_k$ receives at least $n-f$ sets of acceptable values that all contain some value v, then $coord_k$ votes for v (line 11), and sends its vote to all at second round $4k-2$. Upon receiving a vote for v, any process locks v in the current phase (line 18), releases any earlier lock on v, and sends an acknowledgment to $coord_k$ at the next round $4k-1$. If the latter process receives acknowledgments from at least $f+1$ processes, then it decides (line 26). Finally locks are released at round $4k$—for any value v, only the lock from the most recent phase is kept, see line 34—and the set of values *acceptable* to p is updated (lines 35 to 38).

Consensus Algorithm for Byzantine Faults (requires $f < n/3$)

Two algorithms for Byzantine faults are given. The first algorithm assumes *signed messages*, which means that any process can verify the origin of all messages. This fault model is called *Byzantine faults with authentication*. The algorithm has the same phase structure as Algorithm 1. The difference is that (1) messages are signed, and (2) "proofs" are carried by some messages. A proof carried by message m sent by some process p_i in phase k consists of a set of signed messages $sgn_j(m', k)$, prov-

ing that p_i received message (m', k) in phase k from p_j before sending m. A proof is carried by the message send at line 16 and line 30 (Algorithm 1). Any process receiving a message carrying a proof accepts the message and behaves accordingly if—and only if the proof is found valid. The algorithm requires $f < n/3$ (less than a third of the processes are faulty).

The second algorithm does not assume a mechanism for signing messages. Compared to Algorithm 1, the structure of a phase is slightly changed. The problem is related to the vote sent by the coordinator (line 15). Can a Byzantine coordinator fool other processes by not sending the right vote? With signed messages, such a behavior can be detected thanks to the "proofs" carried by messages. A different mechanism is needed in the absence of signature.

The mechanism is a small variation of the *Consistent Broadcast* primitive introduced by Srikanth and Toueg [15]. The broadcast primitive ensures that (1) if a non faulty process broadcasts m, then every non faulty process delivers m, and (2) if some non faulty process delivers m, then all non faulty processes also eventually deliver m. The implementation of this broadcast primitive requires two rounds, which define a *superround*. A phase of the algorithm consists now of three superrounds. The superrounds $3k-2$, $3k-1$, $3k$ mimic rounds $4k-3$, $4k-2$, and $4k-1$ of Algorithm 1, respectively. Lock-release of phase k occurs at the end of superround $3k$, i. e., does not require an additional round, as it does in the two previous algorithms. The algorithm also requires $f < n/3$.

The Special Case of Synchronous Communication

By strengthening the round based computational model, the authors show that synchronous communication allow higher resiliency. More precisely, the paper introduces the model called the *basic round model with signals*, in which upon receiving a signal at round r, every process knows that all the non faulty processes have received the messages that it has sent during round r. At each round after *GSR*, each non faulty process is guaranteed to receive a signal. In this computational model, the authors present three new algorithms tolerating less than n benign faults, $n/2$ Byzantine faults with authentication, and $n/3$ Byzantine faults respectively.

Implementation of the Basic Round Model

The last part of the paper consists of algorithms that simulate the basic round model under various synchrony assumption, for crash faults and Byzantine faults: first with partially synchronous communication and synchronous processes (case 1), second with partially synchronous communication and processes (case 2), and finally with partially synchronous processes and synchronous communication (case 3).

In case 1, the paper first assumes the basic case $\Phi = 1$, i. e., all non faulty process progress exactly at the same speed, which means that they have a common notion of time. Simulating the basic round model is simple in this case. In case 2 processes do not have a common notion of time. The authors handle this case by designing an algorithm for clock synchronization. Then each process uses its private clock to determine its current round. So processes alternate between steps of the clock synchronization algorithm and steps simulating rounds of the basic round model. With synchronous communication (case 3), the authors show that for any type of faults, the so-called basic round model with signals is implementable.

Note that, from the very definition of partial synchrony, the six algorithms share the fundamental property of tolerating message losses, provided they occur during a finite period of time.

Upper Bound for Resiliency

In parallel, the authors exhibit upper bounds for the resiliency degree of Consensus algorithms in each partially synchronous model, according to the type of faults. They show that their Consensus algorithms achieve these upper bounds, and so are optimal with respect to their resiliency degree. These results are summarized in Table 1.

Applications

Availability is one of the key features of critical systems, and is defined as the ratio of the time the system is operational over the total elapsed time. Availability of a system can be increased by replicating its critical components. Two main classes of replication techniques have been considered: *active* replication and *passive* replication. The Consensus problem is at the heart of the implementation of these replication techniques. For example, active replication, also called *state machine replication* [10,14], can be implemented using the group communication primitive called *Atomic Broadcast*, which can be reduced to Consensus [3].

Agreement needs also to be reached in the context of distributed transactions. Indeed, all participants of a distributed transaction need to agree on the output *commit* or *abort* of the transaction. This agreement problem, called *Atomic Commitment*, differs from Consensus in the validity property that connects decision values (*commit* or *abort*) to the initial values (favorable to commit, or de-

Consensus with Partial Synchrony, Table 1
Tight resiliency upper bounds (*P* stands for "process", *C* for "communication"; 0 means "asynchronous", 1/2 means "partially synchronous", and 1 means "synchronous")

	$P = 0 \quad C = 0$	$P = 1/2 \quad C = 1/2$	$P = 1 \quad C = 1/2$	$P = 1/2 \quad C = 1$	$P = 1 \quad C = 1$
Benign	0	$\lceil (n-1)/2 \rceil$	$\lceil (n-1)/2 \rceil$	$n-1$	$n-1$
Authenticated Byzantine	0	$\lceil (n-1)/3 \rceil$	$\lceil (n-1)/3 \rceil$	$\lceil (n-1)/2 \rceil$	$n-1$
Byzantine	0	$\lceil (n-1)/3 \rceil$	$\lceil (n-1)/3 \rceil$	$\lceil (n-1)/3 \rceil$	$\lceil (n-1)/3 \rceil$

manding abort) [9]. In the case decisions are required in all executions, the problem can be reduced to Consensus if the abort decision is acceptable although all processes were favorable to commit, in some restricted failure cases.

Open Problems

A slight modification to each of the algorithms given in the paper is to force a process repeatedly to broadcast the message "Decide v" after it decides v. Then the resulting algorithms share the property that all non faulty processes definitely make a decision within $O(f)$ rounds after GSR, and the constant factor varies between 4 (benign faults) and 12 (Byzantine faults). A question raised by the authors at the end of the paper is whether this constant can be reduced. Interestingly, a positive answer has been given later, in the case of benign faults and $f < n/3$, with a constant factor of 2 instead of 4. This can be achieved with deterministic algorithms, see [4], based on the communication schema of the Rabin randomized Consensus algorithm [13].

The second problem left open is the generalization of this algorithmic approach—namely, the design of algorithms that are always safe and that terminate when a sufficiently long good period occurs—to other fault tolerant distributed problems in partially synchronous systems. The latter point has been addressed for the Atomic Commitment and Atomic Broadcast problems (see Sect. "Applications").

Cross References

- ► Asynchronous Consensus Impossibility
- ► Failure Detectors
- ► Randomization in Distributed Computing

Recommended Reading

1. Bar-Noy, A., Dolev, D., Dwork, C., Strong, H.R.: Shifting Gears: Changing Algorithms on the Fly To Expedite Byzantine Agreement. In: PODC, 1987, pp. 42–51
2. Chandra, T.D., Hadzilacos, V., Toueg, S.: The Weakest Failure Detector for Solving Consensus. J. ACM **43**(4), 685–722 (1996)
3. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43**(2), 225–267 (1996)
4. Charron-Bost, B., Schiper A.: The "Heard-Of" model: Computing in distributed systems with benign failures. Technical Report, EPFL (2007)
5. Dolev, D., Dwork, C., Stockmeyer, L.: On the minimal synchrony needed for distributed consensus. J. ACM **34**(1), 77–97 (1987)
6. Dolev, D., Strong, H.R.: Authenticated Algorithms for Byzantine Agreement. SIAM J. Comput. **12**(4), 656–666 (1983)
7. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
8. Fischer, M., Lynch, N., Paterson, M.: Impossibility of Distributed Consensus with One Faulty Process. J. ACM **32**, 374–382 (1985)
9. Gray, J.: A Comparison of the Byzantine Agreement Problem and the Transaction Commit Problem. In: Fault-Tolerant Distributed Computing [Asilomar Workshop 1986]. LNCS, vol. 448, pp. 10–17. Springer, Berlin (1990)
10. Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System. Commun. ACM **21**(7), 558–565 (1978)
11. Lamport, L.: The Part-Time Parliament. ACM Trans. on Computer Systems **16**(2), 133–169 (1998)
12. Pease, M.C., Shostak, R.E., Lamport, L.: Reaching Agreement in the Presence of Faults. J. ACM **27**(2), 228–234 (1980)
13. Rabin, M.: Randomized Byzantine Generals. In: Proc. 24th Annual ACM Symposium on Foundations of Computer Science, 1983, pp. 403–409
14. Schneider, F.B.: Replication Management using the State-Machine Approach. In Sape Mullender, editor, Distributed Systems, pp. 169–197. ACM Press (1993)
15. Srikanth, T.K., Toueg, S.: Simulating Authenticated Broadcasts to Derive Simple Fault-Tolerant Algorithms. Distrib. Comp. **2**(2), 80–94 (1987)

Constructing a Galled Phylogenetic Network

2006; Jansson, Nguyen, Sung

Wing-Kin Sung
Department of Computer Science, National University of Singapore, Singapore, Singapore

Keywords and Synonyms

Topology with independent recombination events; Galled-tree; Gt-network; Level-1 phylogenetic network

Problem Definition

A *phylogenetic tree* is a binary, rooted, unordered tree whose leaves are distinctly labeled. A *phylogenetic network* is a generalization of a phylogenetic tree formally defined as a rooted, connected, directed acyclic graph in which: (1) each node has outdegree at most 2; (2) each node has indegree 1 or 2, except the root node, which has indegree 0; (3) no node has both indegree 1 and outdegree 1; and (4) all nodes with outdegree 0 are labeled by elements from a finite set L in such a way that no two nodes are assigned the same label. Nodes of outdegree 0 are referred to as *leaves* and identified with their corresponding elements in L. For any phylogenetic network N, let $\mathcal{U}(N)$ be the undirected graph obtained from N by replacing each directed edge by an undirected edge. N is said to be a *galled phylogenetic network* (*galled network* for short) if all cycles in $\mathcal{U}(N)$ are node-disjoint. Galled networks are also known in the literature as *topologies with independent recombination events* [17], *galled trees* [3], *gt-networks* [13], and *level-1 phylogenetic networks* [2,7].

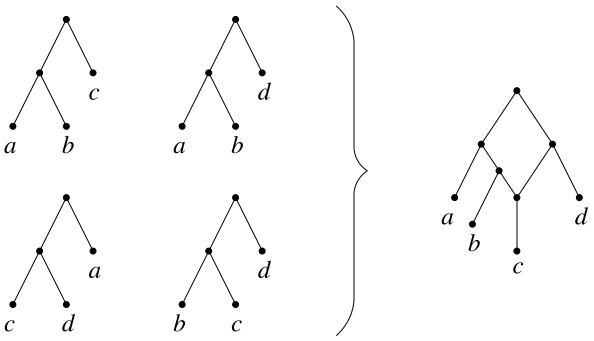
A phylogenetic tree with exactly three leaves is called a *rooted triplet*. The unique rooted triplet on a leaf set $\{x, y, z\}$ in which the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of x and z (or, equivalently, where the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of y and z) is denoted by $(\{x, y\}, z)$. For any phylogenetic network N, a rooted triplet t is said to be *consistent* with N if t is an induced subgraph of N, and a set $\mathcal{T}$ of rooted triplets is *consistent* with N if every rooted triplet in $\mathcal{T}$ is consistent with N.

Denote the set of leaves in any phylogenetic network N by $\Lambda(N)$, and for any set $\mathcal{T}$ of rooted triplets, define $\Lambda(\mathcal{T}) = \bigcup_{t_i \in \mathcal{T}} \Lambda(t_i)$. A set $\mathcal{T}$ of rooted triplets is *dense* if for each $\{x, y, z\} \subseteq \Lambda(\mathcal{T})$ at least one of the three possible rooted triplets $(\{x, y\}, z)$, $(\{x, z\}, y)$, and $(\{y, z\}, x)$ belongs to $\mathcal{T}$. If $\mathcal{T}$ is dense, then $|\mathcal{T}| = \Theta(|\Lambda(\mathcal{T})|^3)$. Furthermore, for any set $\mathcal{T}$ of rooted triplets and $L' \subseteq \Lambda(\mathcal{T})$, define $\mathcal{T} \mid L'$ as the subset of $\mathcal{T}$ consisting of all rooted triplets t with $\Lambda(t) \subseteq L'$. The problem [8] considered here is as follows.

Problem 1 *Given a set $\mathcal{T}$ of rooted triplets, output a galled network N with $\Lambda(N) = \Lambda(\mathcal{T})$ such that N and $\mathcal{T}$ are consistent, if such a network exists; otherwise, output null. (See Fig. 1 for an example.)*

Another related problem is the forbidden triplet problem [4]. It is defined as follows.

Problem 2 *Given two sets $\mathcal{T}$ and $\mathcal{F}$ of rooted triplets, a galled network N $\Lambda(N) = \Lambda(\mathcal{T})$ such that (1) N and $\mathcal{T}$ are consistent and (2) every rooted triplet in $\mathcal{F}$ is not consistent with N. If such a network N exists, it is to be reported; otherwise, output null.*



Constructing a Galled Phylogenetic Network, Figure 1
A dense set $\mathcal{T}$ of rooted triplets with leaf set $\{a, b, c, d\}$ and a galled phylogenetic network which is consistent with $\mathcal{T}$. Note that this solution is not unique

Below, write $L = \Lambda(\mathcal{T})$ and $n = |L|$.

Key Results

Theorem 1 *Given a dense set $\mathcal{T}$ of rooted triplets with leaf set L, a galled network consistent with $\mathcal{T}$ in $O(n^3)$ time can be reported, where $n = |L|$.*

Theorem 2 *Given a nondense set $\mathcal{T}$ of rooted triplets, it is NP-hard to determine if there exists a galled network that is consistent with $\mathcal{T}$. Also, it is NP-hard to determine if there exists a simple phylogenetic network that is consistent with $\mathcal{T}$.*

Below, the problem of returning a galled network N consistent with the maximum number of rooted triplets in $\mathcal{T}$ for any (not necessarily dense) $\mathcal{T}$ is considered. Since Theorem 2 implies that this problem is NP-hard, approximation algorithms are studied. An algorithm is called k-approximable if it always returns a galled network N such that $N(\mathcal{T})/|\mathcal{T}| \geq k$, where $N(\mathcal{T})$ is the number of rooted triplets in $\mathcal{T}$ that are consistent with N.

Theorem 3 *Given a set of rooted triplets $\mathcal{T}$, there is no approximation algorithm that infers a galled network N such that $N(\mathcal{T})/|\mathcal{T}| \geq 0.4883$.*

Theorem 4 *Given a set of rooted triplets $\mathcal{T}$, there exists an approximation algorithm for inferring a galled network N such that $N(\mathcal{T})/|\mathcal{T}| \geq 5/12$. The running time of the algorithm is $O(|\Lambda(\mathcal{T})||\mathcal{T}|^3)$.*

The next theorem considers the forbidden triplet problem.

Theorem 5 *Given two sets of rooted triplets $\mathcal{T}$ and $\mathcal{F}$, there exists an $O(|L|^2|\mathcal{T}|(|\mathcal{T}| + |\mathcal{F}|))$-time algorithm for inferring a galled network N that guarantees $|N(\mathcal{T})| - |N(\mathcal{F})| \geq 5/12(|\mathcal{T}| - |\mathcal{F}|)$.*

Applications

Phylogenetic networks are used by scientists to describe evolutionary relationships that do not fit the traditional models in which evolution is assumed to be treelike (see, e. g., [12,16]). Evolutionary events such as horizontal gene transfer or hybrid speciation (often referred to as *recombination events*) that suggest convergence between objects cannot be represented in a single tree [3,5,13,15,17] but can be modeled in a phylogenetic network as internal nodes having more than one parent. Galled networks are an important type of phylogenetic network that have attracted special attention in the literature [2,3,13,17] due to their biological significance (see [3]) and their simple, almost treelike, structure. When the number of recombination events is limited and most of the recombination events have occurred recently, a galled network may suffice to accurately describe the evolutionary process under study [3].

An open challenge in the field of phylogenetics is to develop efficient and reliable methods for constructing and comparing phylogenetic networks. For example, to construct a meaningful phylogenetic network for a large subset of the human population (which may subsequently be used to help locate regions in the genome associated with some observable trait indicating a particular disease) in the future, efficient algorithms are crucial because the input can be expected to be very large.

The motivation behind the rooted triplet approach taken in this paper is that a highly accurate tree for each cardinality three subset of a leaf set can be obtained through maximum-likelihood-based methods such as [1] or Sibley–Ahlquist-style DNA–DNA hybridization experiments (see [10]). Hence, the algorithms presented in [7] and here can be used as the merging step in a divide-and-conquer approach to constructing phylogenetic networks analogous to the quartet method paradigm for inferring unrooted phylogenetic trees [9,11] and other supertree methods (see [6,14] and references therein). Dense input sets in particular are considered since this case can be solved in polynomial time.

Open Problems

For the rooted triplet problem, the current approximation ratio is not tight ($0.4883 \geq N(\mathcal{T})/|\mathcal{T}| \geq 5/12$). It is open if a tight approximation ratio can be found for this problem. Similarly, a tight approximation ratio needs to be found for the forbidden triplet problem.

Another direction is to work on a fixed-parameter polynomial-time algorithm. Assume the number of hybrid nodes is bounded by h. Can an algorithm that is polynomial in $|\mathcal{T}|$ while exponential in h be given?

Cross References

- ▶ Directed Perfect Phylogeny (Binary Characters)
- ▶ Distance-Based Phylogeny Reconstruction (Fast-Converging)
- ▶ Distance-Based Phylogeny Reconstruction (Optimal Radius)
- ▶ Perfect Phylogeny (Bounded Number of States)
- ▶ Phylogenetic Tree Construction from a Distance Matrix

Recommended Reading

1. Chor, B., Hendy, M., Penny, D.: Analytic solutions for three-taxon ML_{MC} trees with variable rates across sites. In: Proc. 1st Workshop on Algorithms in Bioinformatics (WABI 2001). LNCS, vol. 2149, pp. 204–213. Springer, Berlin (2001)
2. Choy, C., Jansson, J., Sadakane, K., Sung, W.-K.: Computing the maximum agreement of phylogenetic networks. In: Proc. Computing: the 10th Australasian Theory Symposium (CATS 2004), 2004, pp. 33–45
3. Gusfield, D., Eddhu, S., Langley, C.: Efficient reconstruction of phylogenetic networks with constrained recombination. In: Proc. of Computational Systems Bioinformatics (CSB2003), 2003 pp. 363–374
4. He, Y.-J., Huynh, T.N.D., Jannson, J., Sung, W.-K.: Inferring phylogenetic relationships avoiding forbidden rooted triplets. J Bioinform. Comput. Biol. **4**(1), 59–74 (2006)
5. Hein, J.: Reconstructing evolution of sequences subject to recombination using parsimony. Math. Biosci. **98**(2), 185–200 (1990)
6. Henzinger, M.R., King, V., Warnow, T.: Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. Algorithmica **24**(1), 1–13 (1999)
7. Jansson, J., Sung, W.-K.: Inferring a level-1 phylogenetic network from a dense set of rooted triplets. In: Proc. 10th International Computing and Combinatorics Conference (COCOON 2004), 2004
8. Jansson, J., Nguyen, N.B., Sung, W.-K.: Algorithms for combining rooted triplets into a galled phylogenetic network. SIAM J. Comput. **35**(5), 1098–1121 (2006)
9. Jiang, T., Kearney, P., Li, M.: A polynomial time approximation scheme for inferring evolutionary trees from quartet topologies and its application. SIAM J. Comput. **30**(6), 1942–1961 (2001)
10. Kannan, S., Lawler, E., Warnow, T.: Determining the evolutionary tree using experiments. J. Algorithms **21**(1), 26–50 (1996)
11. Kearney, P.: Phylogenetics and the quartet method. In: Jiang, T., Xu, Y., and Zhang, M.Q. (eds.) Current Topics in Computational Molecular Biology, pp. 111–133. MIT Press, Cambridge (2002)

12. Li., W.-H.: Molecular Evolution. Sinauer, Sunderland (1997)
13. Nakhleh, L., Warnow, T., Linder, C.R.: Reconstructing reticulate evolution in species – theory and practice. In: Proc. 8th Annual International Conference on Research in Computational Molecular Biology (RECOMB 2004), 2004, pp. 337–346
14. Ng, M.P., Wormald, N.C.: Reconstruction of rooted trees from subtrees. Discrete Appl. Math. **69**(1–2), 19–31 (1996)
15. Posada, D., Crandall, K.A.: Intraspecific gene genealogies: trees grafting into networks. TRENDS Ecol. Evol. **16**(1), 37–45 (2001)
16. Setubal, J.C., Meidanis, J.: Introduction to Computational Molecular Biology. PWS, Boston (1997)
17. Wang, L., Zhang, K., Zhang, L.: Perfect phylogenetic networks with recombination. J. Comput. Biol. **8**(1), 69–78 (2001)

Coordination Ratio

► Price of Anarchy
► Selfish Unsplittable Flows: Algorithms for Pure Equilibria
► Stackelberg Games: The Price of Optimum

CPU Time Pricing

2005; Deng, Huang, Li

Li-Sha Huang
Department of Compurter Science, Tsinghua University, Beijing, China

Keywords and Synonyms

Competitive auction; Market equilibrium; Resource scheduling

Problem Definition

This problem is concerned with a Walrasian equilibrium model to determine the prices of CPU time. In a market model of a CPU job scheduling problem, the owner of the CPU processing time sells time slots to customers and the prices of each time slot depends on the seller's strategy and the customers' bids (valuation functions). In a Walrasian equilibrium, the market is clear and each customer is most satisfied according to its valuation function and current prices. The work of Deng, Huang, and Li [1] establishes the existence conditions of Walrasion equilibrium, and obtains complexity results to determine the existence of equilibrium. It also discusses the issues of excessive supply of CPU time and price dynamics.

Notations

Consider a combinatorial auction (Ω, I, V):

- Commodities: The seller sells m kinds of indivisible commodities. Let $\Omega = \{\omega_1 \times \delta_1, \ldots, \omega_m \times \delta_m\}$ denote the set of commodities, where δ_j is the available quantity of the item ω_j.
- Agents: There are n agents in the market acting as buyers, denoted by $I = \{1, 2, \ldots, n\}$.
- Valuation functions: Each buyer $i \in I$ has a valuation function $v_i : 2^{\Omega} \to \mathbb{R}^+$ to submit the maximum amount of money he is willing to pay for a certain bundle of items. Let $V = \{v_1, v_2, \ldots, v_n\}$.

An XOR combination of two valuation functions v_1 and v_2 is defined by:

$$(v_1 \text{ XOR } v_2)(S) = \max\{v_1(S), v_2(S)\}$$

An *atomic bid* is a valuation function v denoted by a pair (S, q), where $S \subset \Omega$ and $q \in \mathbb{R}^+$:

$$v(T) = \begin{cases} q\,, & \text{if } S \subset T \\ 0\,, & \text{otherwise} \end{cases}$$

Any valuation function v_i can be expressed by an XOR combination of atomic bids,

$$v_i = (S_{i1}, q_{i1}) \text{ XOR } (S_{i2}, q_{i2}) \ldots \text{ XOR } (S_{in}, q_{in})$$

Given (Ω, I, V) as the input, the seller will determine an *allocation* and a *price vector* as the output:

- An *allocation* $X = \{X_0, X_1, X_2, \ldots, X_n\}$ is a partition of Ω, in which X_i is the bundle of commodities assigned to buyer i and X_0 is the set of unallocated commodities.
- A *price* vector p is a non-negative vector in $\mathbb{R}^m$, whose jth entry is the price of good $\omega_j \in \Omega$.

For any subset $T = \{\omega_1 \times \sigma_1, \ldots, \omega_m \times \sigma_m\} \subset \Omega$, define $p(T)$ by $p(T) = \sum_{j=1}^{m} \sigma_j p_j$. If buyer i is assigned to a bundle X_i, his *utility* is $u_i(X_i, p) = v_i(X_i) - p(X_i)$.

Definition A *Walrasian equilibrium* for a combinatorial auction (Ω, I, V) is a tuple (X, p), where $X = \{X_0, X_1, \ldots, X_n\}$ is an allocation and p is a price vector, satisfying that:

(1) $p(X_0) = 0$;

(2) $u_i(X_i, p) \geq u_i(B, p), \quad \forall B \subset \Omega, \quad \forall 1 \leq i \leq n$

Such a price vector is also called a market clearing price, or Walrasian price, or equilibrium price.

The CPU Job-Scheduling Problem

There are two types of players in a market-driven CPU resource allocation model: a resource provider and n consumers. The provider sells to the consumers CPU time

slots and the consumers each have a job that requires a fixed number of CPU time, and its valuation function depends on the time slots assigned to the job, usually the last assigned CPU time slot. Assume that all jobs are released at time $t = 0$ and the ith job needs s_i time units. The jobs are interruptible without preemption cost, as is often modeled for CPU jobs.

Translating into the language of combinatorial auctions, there are m commodities (time units), $\Omega = \{\omega_1, \ldots, \omega_m\}$, and n buyers (jobs) , $I = \{1, 2, \ldots, n\}$, in the market. Each buyer has a valuation function v_i, which only depends on the completion time. Moreover, if not explicitly mentioned, every job's valuation function is non-increasing w.r.t. the completion time.

Key Results

Consider the following linear programming problem:

$$\max \sum_{i=1}^{n} \sum_{j=1}^{k_i} q_{ij} x_{ij}$$

$$\text{s.t.} \sum_{i,j|\omega_k \in S_{ij}} x_{ij} \leq \delta_k \,, \quad \forall \omega_k \in \Omega$$

$$\sum_{j=1}^{r_i} x_{ij} \leq 1 \,, \quad \forall 1 \leq i \leq n$$

$$0 \leq x_{ij} \leq 1 \,, \quad \forall i, j$$

Denote the problem by **LPR** and its integer restriction by **IP**. The following theorem shows that a non-zero gap between the integer programming problem **IP** and its linear relaxation implies the non-existence of the Walrasian equilibrium.

Theorem 1 *In a combinatorial auction, the Walrasian equilibrium exists if and only if the optimum of **IP** equals the optimum of **LPR**. The size of the LP problem is linear to the total number of XOR bids.*

Theorem 2 *Determination of the existence of Walrasian equilibrium in a CPU job scheduling problem is strong NP-hard.*

Now consider a job scheduling problem in which the customers' valuation functions are all linear. Assume n jobs are released at the time $t = 0$ for a single machine, the jth job's time span is $s_j \in \mathbb{N}^+$ and weight $w_j \geq 0$. The goal of the scheduling is to minimize the weighted completion time: $\sum_{i=1}^{n} w_i t_i$, where t_i is the completion time of job i. Such a problem is called an MWCT (Minimal Weighted Completion Time) problem.

Theorem 3 *In a single-machine MWCT job scheduling problem, Walrasian equilibrium always exists when $m \geq EM + \Delta$, where m is the total number of processor time, $EM = \sum_{i=1}^{n} s_i$ and $\Delta = \max_k \{s_k\}$. The equilibrium can be computed in polynomial time.*

The following theorem shows the existence of a non-increasing price sequence if Walrasian equilibrium exists.

Theorem 4 *If there exists a Walrasian equilibrium in a job scheduling problem, it can be adjusted to an equilibrium with consecutive allocation and a non-increasing equilibrium price vector.*

Applications

Information technology has changed people's lifestyles with the creation of many digital goods, such as word processing software, computer games, search engines, and online communities. Such a new economy has already demanded many theoretical tools (new and old, of economics and other related disciplines) be applied to their development and production, marketing, and pricing. The lack of a full understanding of the new economy is mainly due to the fact that digital goods can often be re-produced at no additional cost, though multi-fold other factors could also be part of the difficulty. The work of Deng, Huang, and Li [1] focuses on CPU time as a product for sale in the market, through the Walrasian pricing model in economics. CPU time as a commercial product is extensively studied in grid computing. Singling out CPU time pricing will help us to set aside other complicated issues caused by secondary factors, and a complete understanding of this special digital product (or service) may shed some light on the study of other goods in the digital economy.

The utilization of CPU time by multiple customers has been a crucial issue in the development of operating system concept. The rise of grid computing proposes to fully utilize computational resources, e. g. CPU time, disk space, bandwidth. Market-oriented schemes have been proposed for efficient allocation of computational grid recourses, by [2,5]. Later, various practical and simulation systems have emerged in grid resource management. Besides the resource allocation in grids, an economic mechanism has also been introduced to TCP congestion control problems, see Kelly [4].

Cross References

- ► Adwords Pricing
- ► Competitive Auction
- ► Incentive Compatible Selection
- ► Price of Anarchy

Recommended Reading

1. Deng, X., Huang, L.-S., Li, M.: On Walrasian Price of CPU time. In: Proceedings of COCOON'05, Knming, 16–19 August 2005, pp. 586–595. Algorithmica **48**(2), 159–172 (2007)
2. Ferguson, D., Yemini, Y., Nikolaou, C.: Microeconomic Algorithms for Load Balancing in Distributed Computer Systems. In: Proceedings of DCS'88, pp. 419–499. San Jose, 13–17 June 1988,
3. Goldberg, A.V., Hartline, J.D., Wright, A.: Competitive Auctions and Digital Goods. In: Proceedings of SODA'01, pp. 735–744. Washington D.C., 7–9 January 2001
4. Kelly, F.P.: Charging and rate control for elastic traffic. Eur. Trans. Telecommun. **8**, 33–37 (1997)
5. Kurose, J.F., Simha, R.: A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems. IEEE Trans. Comput. **38**(5), 705–717 (1989)
6. Nisan, N.: Bidding and Allocation in Combinatorial Auctions. In: Proceedings of EC'00, pp. 1–12. Minneapolis, 17–20 October 2000

Critical Range for Wireless Networks

2004; Wan, Yi

Chih-Wei Yi
Department of Computer Science,
National Chiao Tung University, Hsinchu City, Taiwan

Keywords and Synonyms

Random geometric graphs; Monotonic properties; Isolated nodes; Connectivity; Gabriel graphs; Delaunay triangulations; Greedy forward routing

Problem Definition

Given a point set V, a graph of the vertex set V in which two vertices have an edge if and only if the distance between them is at most r for some positive real number r is called a r-disk graph over the vertex set V and denoted by $G_r(V)$. If $r_1 \leq r_2$, obviously $G_{r_1}(V) \subseteq G_{r_2}(V)$. A graph property is monotonic (increasing) if a graph is with the property, then every supergraph with the same vertex set also has the property. The critical-range problem (or critical-radius problem) is concerned with the minimal range r such that $G_r(V)$ is with some monotonic property. For example, graph connectivity is monotonic and crucial to many applications. It is interesting to know whether $G_r(V)$ is connected or not. Let $\rho_{\text{con}}(V)$ denote the minimal range r such that $G_r(V)$ is connected. Then, $G_r(V)$ is connected if $r \geq \rho_{\text{con}}(V)$, and otherwise not connected. Here $\rho_{\text{con}}(V)$ is called the critical range for connectivity of V. Formally, the critical-range problem is defined as follows.

Definition 1 The critical range for a monotonic graph property π over a point set V, denoted by $\rho_\pi(V)$, is the smallest range r such that $G_r(V)$ has property π.

From another aspect, for a given geometric property, a corresponding geometric structure is usually embedded. In many cases, the critical-range problem for graph properties is related or equivalent to the longest-edge problem of corresponding geometric structures. For example, if $G_r(V)$ is connected, it contains a Euclidean minimal spanning tree (EMST), and $\rho_{\text{con}}(V)$ is equal to the largest edge length of the EMST. So the critical range for connectivity problem is equivalent to the longest edge of the EMST problem, and the critical range for connectivity is the smallest r such that $G_r(V)$ contains the EMST.

In most cases, given an instance, the critical range can be calculated by polynomial time algorithms. So it is not a hard problem to decide the critical range. Researchers are interested in the probabilistic analysis of the critical range, especially asymptotic behaviors of r-disk graphs over random point sets. Random geometric graphs [8] is a general term for the theory about r-disk graphs over random point sets.

Key Results

In the following, problems are discussed in a 2D plane. Let $X_1, X_2, \cdots$ be independent and uniformly distributed random points on a bounded region A. Given a positive integer n, the point process $\{X_1, X_2, \ldots, X_n\}$ is referred to as the uniform n-point process on A, and is denoted by $\mathcal{X}_n(A)$. Given a positive number λ, let $Po(\lambda)$ be a Poisson random variable with parameter λ, independent of $\{X_1, X_2, \ldots\}$. Then the point process $\{X_1, X_2, \ldots, X_{Po(n)}\}$ is referred to as the Poisson point process with mean n on A, and is denoted by $\mathcal{P}_n(A)$. A is called a deployment region. An event is said to be asymptotic almost sure if it occurs with a probability that converges to 1 as $n \to \infty$.

In a graph, a node is "isolated" if it has no neighbor. If a graph is connected, there exists no isolated node in the graph. The asymptotic distribution of the number of isolated nodes is given by the following theorem [2,6,14].

Theorem 1 *Let* $r_n = \sqrt{\frac{\ln n + \xi}{\pi n}}$ *and* Ω *be a unit-area disk or square. The number of isolated nodes in* $G_r(\mathcal{X}_n(\Omega))$ *or* $G_r(\mathcal{P}_n(\Omega))$ *is asymptotically Poisson with mean* $e^{-\xi}$.

According to the theorem, the probability of the event that there is no isolated node is asymptotically equal to $\exp\left(-e^{-\xi}\right)$. In the theory of random geometric graphs, if

a graph has no isolated node, it is almost surely connected. Thus, the next theorem follows [6,8,9].

Theorem 2 *Let* $r_n = \sqrt{\frac{\ln n + \xi}{\pi n}}$ *and* Ω *be a unit-area disk or square. Then,*

$$\Pr\left[G_r\left(\mathcal{X}_n(\Omega)\right) \text{ is connected}\right] \to \exp\left(-e^{-\xi}\right), \text{ and}$$
$$\Pr\left[G_r\left(\mathcal{P}_n(\Omega)\right) \text{ is connected}\right] \to \exp\left(-e^{-\xi}\right).$$

In wireless sensor networks, the deployment region is k-covered if every point in the deployment region is within the coverage ranges of at least k sensors (vertices). Assume the coverage ranges are disks of radius r centered at the vertices. Let k be a fixed non-negative integer, and Ω be the unit-area square or disk centered at the origin $\mathbf{o}$. For any real number t, let $t\Omega$ denote the set $\{tx : x \in \Omega\}$, i. e., the square or disk of area t^2 centered at the origin. Let $C_{n,r}$ (respectively, $C'_{n,r}$) denote the event that Ω is $(k+1)$-covered by the (open or closed) disks of radius r centered at the points in $\mathcal{P}_n(\Omega)$ (respectively, $\mathcal{X}_n(\Omega)$). Let $K_{s,n}$ (respectively, $K'_{s,n}$) denote the event that $\sqrt{s}\Omega$ is $(k+1)$-covered by the unit-area (closed or open) disks centered at the points in $\mathcal{P}_n(\sqrt{s}\Omega)$ (respectively, $\mathcal{X}_n(\sqrt{s}\Omega)$). To simplify the presentation, let η denote the peripheral of Ω, which is equal to 4 (respectively, $2\sqrt{\pi}$) if Ω is a square (respectively, disk). For any $\xi \in \mathbb{R}$, let

$$\alpha(\xi) = \begin{cases} \frac{\left(\frac{\sqrt{\pi}\eta}{2} + e^{-\frac{\xi}{2}}\right)^2}{16\left(2\sqrt{\pi}\eta + e^{-\frac{\xi}{2}}\right)} e^{-\frac{\xi}{2}}, & \text{if } k = 0; \\ \frac{\sqrt{\pi}\eta}{2^{k+6}(k+2)!} e^{-\frac{\xi}{2}}, & \text{if } k \geq 1. \end{cases}$$

and

$$\beta(\xi) = \begin{cases} 4e^{-\xi} + 2\left(\sqrt{\pi} + \frac{1}{\sqrt{\pi}}\right)\eta e^{-\frac{\xi}{2}}, & \text{if } k = 0; \\ \frac{\sqrt{\pi} + \frac{1}{\sqrt{\pi}}}{2^{k-1}k!}\eta e^{-\frac{\xi}{2}}, & \text{if } k \geq 1. \end{cases}$$

The asymptotics of $\Pr\left[C_{n,r}\right]$ and $\Pr\left[C'_{n,r}\right]$ as n approaches infinity, and the asymptotics of $\Pr\left[K_{s,n}\right]$ and $\Pr\left[K'_{s,n}\right]$ as s approaches infinity are given in the following two theorems [4,10,16].

Theorem 3 *Let* $r_n = \sqrt{\frac{\ln n + (2k+1)\ln\ln n + \xi_n}{\pi n}}$. *If* $\lim_{n\to\infty} \xi_n = \xi$ *for some* $\xi \in \mathbb{R}$, *then*

$$1 - \beta(\xi) \leq \lim_{n\to\infty} \Pr\left[C_{n,r_n}\right] \leq \frac{1}{1+\alpha(\xi)}, \text{ and}$$
$$1 - \beta(\xi) \leq \lim_{n\to\infty} \Pr\left[C'_{n,r_n}\right] \leq \frac{1}{1+\alpha(\xi)}.$$

If $\lim_{n\to\infty} \xi_n = \infty$, *then*

$$\lim_{n\to\infty} \Pr\left[C_{n,r_n}\right] = \lim_{n\to\infty} \Pr\left[C'_{n,r_n}\right] = 1.$$

If $\lim_{n\to\infty} \xi_n = -\infty$, *then*

$$\lim_{n\to\infty} \Pr\left[C_{n,r_n}\right] = \lim_{n\to\infty} \Pr\left[C'_{n,r_n}\right] = 0.$$

Theorem 4 *Let* $\mu(s) = \ln s + 2(k+1)\ln\ln s + \xi(s)$. *If* $\lim_{s\to\infty} \xi(s) = \xi$ *for some* $\xi \in \mathbb{R}$, *then*

$$1 - \beta(\xi) \leq \lim_{s\to\infty} \Pr\left[K_{s,\mu(s)s}\right] \leq \frac{1}{1+\alpha(\xi)}, \text{ and}$$
$$1 - \beta(\xi) \leq \lim_{s\to\infty} \Pr\left[K'_{s,\mu(s)s}\right] \leq \frac{1}{1+\alpha(\xi)}.$$

If $\lim_{s\to\infty} \xi(s) = \infty$, *then*

$$\lim_{s\to\infty} \Pr\left[K_{s,\mu(s)s}\right] = \lim_{s\to\infty} \Pr\left[K'_{s,\mu(s)s}\right] = 1.$$

If $\lim_{s\to\infty} \xi(s) = -\infty$, *then*

$$\lim_{s\to\infty} \Pr\left[K_{s,\mu(s)s}\right] = \lim_{s\to\infty} \Pr\left[K'_{s,\mu(s)s}\right] = 0.$$

In Gabriel graphs (GG), two nodes have an edge if and only if there is no other node in the disk using the segment of these two nodes as its diameter. If V is a point set and l is a positive real number, we use $\rho_{\mathrm{GG}}(V)$ to denote the largest edge length of the GG over V, and $N(V, l)$ denotes the number of GG edges over V whose length is at least l. Wan and Yi (2007) [11] gave the following theorem.

Theorem 5 *Let* Ω *be a unit-area disk. For any constant* ξ, $N\left(\mathcal{P}_n(\Omega), 2\sqrt{\frac{\ln n + \xi}{\pi n}}\right)$ *is asymptotically Poisson with mean* $2e^{-\xi}$, *and*

$$\lim_{n\to\infty} \Pr\left[\rho_{GG}\left(\mathcal{P}_n(\Omega)\right) < 2\sqrt{\frac{\ln n + \xi}{\pi n}}\right] = \exp\left(-2e^{-\xi}\right).$$

Let $\rho_{\mathrm{Del}}(V)$ denote the largest edge length of the Delaunay triangulation over a point set V. The following theorem is given by Kozma et al. [3].

Theorem 6 *Let* Ω *be a unit-area disk. Then,*

$$\rho_{Del}\left(\mathcal{X}_n(\Omega)\right) = O\left(\sqrt[3]{\frac{\ln n}{n}}\right).$$

In wireless networks with greedy forward routing (GFR), each node discards a packet if none of its neighbors is

closer to the destination of the packet than itself, or otherwise forwards the packet to the neighbor that is the closest to the destination. Since each node only needs to maintain the locations of its one-hop neighbors and each packet should contain the location of the destination node, GFR can be implemented in a localized and memoryless manner. Because of the existence of local minima where none of the neighbors is closer to the destination than the current node, a packet may be discarded before it reaches its destination. To ensure that every packet can reach its destination, all nodes should have sufficiently large transmission radii to avoid the existence of local minima. Applying the r-disk model, we assume every node has the same transmission radius r, and each pair of nodes with distance at most r has a link. For a point set V, the *critical transmission radius* for GFR is given by

$$\rho_{\mathrm{GFR}}(V) = \max_{(u,v)\in V^2, u\neq v} \left(\min_{\|w-v\|<\|u-v\|} \|w-u\| \right).$$

In the definition, (u, v) is a source–destination pair and w is a node that is closer to v than u. If every node is with a transmission radius not less than $\rho_{\mathrm{GFR}}(V)$, GFR can guarantee the deliverability between any source–destination pair [12].

Theorem 7 *Let Ω be a unit-area convex compact region with bounded curvature, and $\beta_0 = 1/\left(2/3 - \sqrt{3}/2\pi\right) \approx 1.6^2$. Suppose that $n\pi r_n^2 = (\beta + o(1)) \ln n$ for some $\beta > 0$. Then,*

1. *If $\beta > \beta_0$, then $\rho_{GFR}(\mathcal{P}_n(\Omega)) \leq r_n$ is asymptotically almost sure.*
2. *If $\beta < \beta_0$, then $\rho_{GFR}(\mathcal{P}_n(\Omega)) > r_n$ is asymptotically almost sure.*

Applications

In the literature, r-disk graphs (or unit disk graphs by proper scaling) are widely used to model homogeneous wireless networks in which each node is equipped with an omnidirectional antenna. According to the path loss of radio frequency, the transmission ranges (radii) of wireless devices depend on transmission powers. For simplicity, the power assignment problem usually is modeled by a corresponding transmission range assignment problem. Recently, wireless ad-hoc networks have attracted attention from a lot of researchers because of various possible applications. In many of the possible applications, since wireless devices are powered by batteries, transmission range assignment has become one of the most important tools for prolonging system lifetime. By applying the theory of critical ranges, a randomly deployed wireless ad-hoc network may have good properties in high probability if the transmission range is larger than some critical value.

One application of critical ranges is to connectivity of networks. A network is k-vertex-connected if there exist k node-disjoint paths between any pair of nodes. With such a property, at least k distinct communication paths exist between any pair of nodes, and the network is connected even if $k-1$ nodes fail. Thus, with a higher degree of connectivity, a network may have larger bandwidth and higher fault tolerance capacity. In addition, in [9,14], and [15], networks with node or link failures were considered.

Another application is in topology control. To efficiently operate wireless ad-hoc networks, subsets of network topology will be constructed and maintained. The related topics are called topology control. A spanner is a subset of the network topology in which the minimal total cost of a path between any pair of nodes, e. g. distance or energy consumption, is only a constant fact larger than the minimal total cost in the original network topology. Hence spanners are good candidates for virtual backbones. Geometric structures, including Euclidean minimal spanning trees, relative neighbor graphs, Gabriel graphs, Delaunay triangulations, Yao's graphs, etc., are widely used ingredients to construct spanners [1,5,13]. By applying the knowledge of critical ranges, the complexity of algorithm design can be reduced, e. g. [3,11].

Open Problems

A number of problems related to critical ranges remain open. Most problems discussed here apply 2-D plane geometry. In other words, the point set is in the plane. The first direction for future work is to study those problems in high-dimension spaces. Another open research area is on the longest-edge problems for other geometric structures, e. g. relative neighbor graphs and Yao's graphs. A third direction for future work involves considering relations between graph properties. A well-known result in random geometric graphs is that vanishment of isolated nodes asymptotically implies connectivity of networks. But for the wireless networks with unreliable links, this property is still open. In addition, in wireless sensor networks, the relations between connectivity and coverage are also interesting.

Cross References

- ► Applications of Geometric Spanner Networks
- ► Connected Dominating Set
- ► Dilation of Geometric Networks
- ► Geometric Spanners

- ▶ Minimum Geometric Spanning Trees
- ▶ Minimum k-Connected Geometric Networks
- ▶ Randomized Broadcasting in Radio Networks
- ▶ Randomized Gossiping in Radio Networks

Recommended Reading

1. Cartigny, J., Ingelrest, F., Simplot-Ryl, D., Stojmenovic, I.: Localized LMST and RNG based minimum-energy broadcast protocols in ad hoc networks. Ad Hoc Netw. **3**(1), 1–16 (2004)
2. Dette, H., Henze, N.: The limit distribution of the largest nearest-neighbour link in the unit d-cube. J. Appl. Probab. **26**, 67–80 (1989)
3. Kozma, G., Lotker, Z., Sharir, M., Stupp, G.: Geometrically aware communication in random wireless networks. In: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing, 25–28 July 2004, pp. 310–319
4. Kumar, S., Lai, T.H., Balogh, J.: On k-coverage in a mostly sleeping sensor network. In: Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (MobiCom'04), 26 Sept–1 Oct 2004
5. Li, N., Hou, J.C., Sha, L.: Design and analysis of a MST-based distributed topology control algorithm for wireless ad-hoc networks. In: 22nd Annual Joint Conference Of The IEEE Computer And Communications Societies (INFOCOM 2003), vol. 3, 1–3 April 2003, pp. 1702–1712
6. Penrose, M.: The longest edge of the random minimal spanning tree. Ann. Appl. Probab. **7**(2), 340–361 (1997)
7. Penrose, M.: On k-connectivity for a geometric random graph. Random. Struct. Algorithms **15**(2), 145–164 (1999)
8. Penrose, M.: Random Geometric Graphs. Oxford University Press, Oxford (2003)
9. Wan, P.-J., Yi, C.-W.: Asymptotic critical transmission ranges for connectivity in wireless ad hoc networks with Bernoulli nodes. In: IEEE Wireless Communications and Networking Conference (WCNC 2005), 13–17 March 2005
10. Wan, P.-J., Yi, C.-W.: Coverage by randomly deployed wireless sensor networks. In: Proceedings of the 4th IEEE International Symposium on Network Computing and Applications (NCA 2005), 27–29 July 2005
11. Wan, P.-J., Yi, C.-W.: On the longest edge of Gabriel graphs in wireless ad hoc networks. Trans. Parallel Distrib. Syst. **18**(1), 1–16 (2007)
12. Wan, P.-J., Yi, C.-W., Yao, F., Jia, X.: Asymptotic critical transmission radius for greedy forward routing in wireless ad hoc networks. In: Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing, 22–25 May 2006, pp. 25–36
13. Wang, Y., Li, X.-Y.: Localized construction of bounded degree and planar spanner for wireless ad hoc networks, In: Proceedings of the 2003 joint workshop on Foundations of mobile computing (DIALM-POMC'03), 19 Sept 2003, pp. 59–68
14. Yi, C.-W., Wan, P.-J., Li, X.-Y., Frieder, O.: Asymptotic distribution of the number of isolated nodes in wireless ad hoc networks with Bernoulli nodes. In: IEEE Wireless Communications and Networking Conference (WCNC 2003), March 2003
15. Yi, C.-W., Wan, P.-J., Lin, K.-W., Huang, C.-H.: Asymptotic distribution of the Number of isolated nodes in wireless ad hoc networks with unreliable nodes and links. In: the 49th Annual IEEE GLOBECOM Technical Conference (GLOBECOM 2006), 27 Nov–1 Dec 2006
16. Zhang, H., Hou, J.: On deriving the upper bound of α-lifetime for large sensor networks. In: Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc 2004), 24–26 March 2004

Cryptographic Hardness of Learning

1994; Kearns, Valiant

Adam Klivans
Department of Computer Science, University of Texas at Austin, Austin, TX, USA

Keywords and Synonyms

Representation-independent hardness for learning

Problem Definition

This paper deals with proving negative results for distribution-free PAC learning. The crux of the problem is proving that a polynomial-time algorithm for learning various concept classes in the PAC model implies that several well-known cryptosystems are insecure. Thus, if we assume a particular cryptosystem is secure we can conclude that it is impossible to efficiently learn a corresponding set of concept classes.

PAC Learning

We recall here the PAC learning model. Let C be a concept class (a set of functions over n variables), and let D be a distribution over the input space $\{0,1\}^n$. With C we associate a size function size that measures the complexity of each $c \in C$. For example if C is a class of Boolean circuits then size(c) is equal to the number of gates in c. Let A be a randomized algorithm that has access to an oracle which returns labeled examples $(x, c(x))$ for some unknown $c \in C$; the examples x are drawn according to D. Algorithm A PAC-learns concept class C by hypothesis class H if for any $c \in C$, for any distribution D over the input space, and any $\epsilon, \delta > 0$, A runs in time $poly(n, 1/\epsilon, 1/\delta, size(c))$ and produces a hypothesis $h \in H$ such that with probability at least $(1-\delta)$, $Pr_D[c(x) \neq h(x)] < \epsilon$. This probability is taken over the random coin tosses of A as well as over the random labeled examples seen from distribution D. When $H = C$ (the hypothesis must be some concept in C) then A is a *proper* PAC learning algorithm. In this article is not assumed $H = C$, i. e. hardness results for *representation-independent* learning algorithms are discussed. The only

assumption made on H is that for each $h \in H$, h can be evaluated in polynomial time for every input of length n.

Cryptographic Primitives

Also required is knowledge of various cryptographic primitives such as public-key cryptosystems, one-way functions, and one-way trapdoor functions. For a formal treatment of these primitives refer to Goldreich [3].

Informally, a function f is *one-way* if, after choosing a random x of length n and giving an adversary A only $f(x)$, it is computationally intractable for A to find y such that $f(y) = f(x)$. Furthermore, given x, $f(x)$ can be evaluated in polynomial time. That is, f is easy to compute one-way, but there is no polynomial-time algorithm for finding pre-images of f on randomly chosen inputs. Say a function f is *trapdoor* if f is one-way, but if an adversary A is given access to a secret "trapdoor" d, then A *can* efficiently find random pre-images of f.

Trapdoor functions that are permutations are closely related to *public-key cryptosystems*: imagine a person Alice who wants to allow others to secretly communicate with her. She publishes a one-way trapdoor permutation f so that it is publicly available to everyone, but keeps the "trapdoor" d to herself. Then Bob can send Alice a secret message x by sending her $f(x)$. Only Alice is able to invert f (recall f is a permutation) and recover x because only she knows d.

Key Results

The main insight in Kearns and Valiant's work is the following: if f is a trapdoor one-way function, and C is a circuit class containing the set of functions capable of inverting f given access to the trap-door, then C is not efficiently PAC learnable. I. e., assuming the difficulty of inverting trap-door function f, there is a distribution on $\{0,1\}^n$ where no learning algorithm can succeed in learning f's associated decryption function.

The following theorem is stated in the (closely related) language of public-key cryptosystems:

Theorem 1 (cryptography and learning; cf. Kearns & Valiant[4]) *Consider a public-key cryptosystem for encrypting individual bits into n-bit strings. Let C be a concept class that contains all the decryption functions* $\{0,1\}^n \to \{0,1\}$ *of the cryptosystem. If C is PAC-learnable in polynomial time then there is a polynomial-time distinguisher between the encryptions of 0 and 1.*

The intuition behind the proof is as follows: fix an encryption function f, associated secret key d, and let C be a class of functions such that the problem of inverting $f(x)$ given d can be computed by an element c of C; notice that knowledge of d is not necessary to generate a polynomial-size sample of $(x, f(x))$ pairs.

If C is PAC learnable, then given a relatively small number of encrypted messages $(x, f(x))$, a learning algorithm A can find a hypothesis h that will approximate c and thus have a non-negligible advantage for decrypting future *randomly* encrypted messages. This violates the security properties of the cryptosystem.

A natural question follows: "what is the simplest concept class that can compute the decryption function for secure public-key cryptosystems?" For example, if a public-key cryptosystem is proven to be secure, and encrypted messages can be decrypted (given the secret key) by polynomial-size DNF formulas, then, by Theorem 1, one could conclude that polynomial-size DNF formulas cannot be learned in the PAC model.

Kearns and Valiant do not obtain such a hardness result for learning DNF formulas (it is still an outstanding open problem), but they do obtain a variety of hardness results assuming the security of various well-known public-key cryptosystems based on the hardness of number-theoretic problems such as factoring.

The following list summarizes their main results:

- Let C be the class of polynomial-size Boolean formulas (not necessarily DNF formulas) or polynomial-size circuits of logarithmic depth. Assuming that the RSA cryptosystem is secure, or recognizing quadratic residues is intractable, or factoring Blum integers is intractable, C is not PAC learnable.
- Let C be the class of polynomial-size deterministic finite automata. Under the same assumptions as above, C is not PAC learnable.
- Let C be the class of constant depth threshold circuits of polynomial size. Under the same assumptions as above, C is not PAC learnable. The depth of the circuit class is not specified but it can be seen to be at most 4.

Kearns and Valiant also prove the intractability of finding optimal solutions to related coloring problems assuming the security of the above cryptographic primitives (breaking RSA, for example).

Relationship to Hardness Results for Proper Learning

The key results above should not be confused with the extensive literature regarding hardness results for *properly* PAC learning concept classes. For example, it is known [1] that, unless RP=NP, it is impossible to properly PAC learn polynomial-size DNF formulas (i. e., require the learner to learn DNF formulas by outputting a DNF formula as its hypothesis). Such results are *incomparable* to the work

of Kearns and Valiant, as they require something much stronger from the learner but take a much weaker assumption (RP=NP is a weaker assumption than the assumption that RSA is secure).

Applications and Related Work

Valiant [10] was the first to observe that the existence of a particular cryptographic primitive (pseudorandom functions) implies hardness results for PAC learning concept classes. The work of Kearns and Valiant has subsequently found many applications. Klivans and Sherstov have recently shown [7] that the problem of PAC learning intersections of halfspaces (a very simple depth-2 threshold circuit) is intractable unless certain lattice-based cryptosystems due to Regev [9] are not secure. Their result makes use of the Kearns and Valiant approach. Angluin and Kharitonov [2] have extended the Kearns and Valiant paradigm to give cryptographic hardness results for learning concept classes even if the learner has *query access* to the unknown concept. Kharitonov [6] has given hardness results for learning polynomial-size, constant depth circuits that assumes the existence of secure pseudorandom generators rather than the existence of public-key cryptosystems.

Open Problems

The major open problem in this line of research is to prove a cryptographic hardness result for PAC learning polynomial-size DNF formulas. Currently, polynomial-size DNF formulas seem far too weak to compute cryptographic primitives such as the decryption function for a well-known cryptosystem. The fastest known algorithm for PAC learning DNF formulas runs in time $2^{\tilde{O}(n^{1/3})}$ [8].

Cross References

► PAC Learning

Recommended Reading

1. Alekhnovich, M., Braverman, M., Feldman, V., Klivans, A. R., Pitassi, T.: Learnability and automatizability. In: Proceedings of the 45th Symposium on Foundations of Computer Science, 2004
2. Angluin, D., Kharitonov, M.: When Won't Membership Queries Help? J. Comput. Syst. Sci. **50**, (1995)
3. Goldreich, O.: Foundations of Cryptography: Basic Tools. Cambridge University Press (2001)
4. Kearns, M., Valiant, L.: Cryptographic limitations on learning Boolean formulae and finite automata. J. ACM **41**(1), 67–95 (1994)
5. Kearns, M., Vazirani, U.: An introduction to computational learning theory. MIT Press, Cambridge (1994)
6. Kharitonov, M.: Cryptographic hardness of distribution-specific learning. In: Proceedings of the Twenty-Fifth Annual Symposium on Theory of Computing, 1993, pp. 372–381
7. Klivans, A. , Sherstov, A. A.: Cryptographic Hardness for Learning Intersections of Halfspaces. In: Proceedings of the 47th Symposium on Foundations of Computer Science, 2006
8. Klivans, A., Servedio, R.: Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. In: Proceedings of the 33rd Annual Symposium on Theory of Computing, 2001
9. Regev, O.: New Lattice-Based Cryptographic Constructions. J. ACM **51**, 899–942 (2004)
10. Valiant, L.: A theory of the learnable. Commun. ACM **27**(11), 1134–1142 (1984)

Cuckoo Hashing

2001; Pagh, Rodler

RASMUS PAGH
Computational Logic and Algorithms Group,
IT University of Copenhagen, Copenhagen, Denmark

Problem Definition

A *dictionary* (also known as an *associative array*) is an abstract data structure capable of storing a set S of elements, referred to as *keys*, and information associated with each key. The operations supported by a dictionary are insertion of a key (and associated information), deletion of a key, and lookup of a key (retrieving the associated information). In case a lookup is made on a key that is not in S, this must be reported by the data structure.

Hash tables is a class of data structures used for implementing dictionaries in the RAM model of computation. *Open addressing* hash tables is a particularly simple type of hash tables, where the data structure is an array such that each entry either contains a key of S or is marked "empty". *Cuckoo hashing* addresses the problem of implementing an open addressing hash table with *worst case* constant lookup time. Specifically, a constant number of entries in the hash table should be associated with each key x, such that x is present in one of these entries if $x \in S$.

In the following it is assumed that a key as well as the information associated with a key are single machine words. This is essentially without loss of generality: If more associated data is wanted, it can be referred to using a pointer. If keys are longer than one machine word, they can be mapped down to a single (or a few) machine words using universal hashing [3], and the described method used on the hash values (which are unique to each key with high probability). The original key must then be stored as associated data. Let n denote an upper bound on the size

of S. To allow the size of the set to grow beyond n, *global rebuilding* can be used.

Key Results

Prehistory

It has been known since the advent of universal hashing [3] that if the hash table has $r \geq n^2$ entries, a lookup can be implemented by retrieving just a single entry in the hash table. This is done by storing a key x in entry $h(x)$ of the hash table, where h is a function from the set of machine words to $\{1, \ldots, n^2\}$. If h is chosen at random from a *universal family* of hash functions [3] then with probability at least 1/2 every key in S is assigned a unique entry. The same behavior would be seen if h was a random function, but in contrast to random functions there are universal families that allow efficient storage and evaluation of h (constant number of machine words, and constant evaluation time).

This overview concentrates on the case where the space used by the hash table is linear, $r = O(n)$. It was shown by Azar et al. [1] that it is possible to combine linear space with worst case constant lookup time. It was not considered how to construct the data structure. Since randomization is used, all schemes discussed have a probability of error. However, this probability is small, $O(1/n)$ or less for all schemes, and an error can be handled by *rehashing* (changing the hash functions and rebuilding the hash table). The result of [1] was shown under the assumption that the algorithm is given free access to a number of truly random hash functions. In many of the subsequent papers it is shown how to achieve the bounds using explicitly defined hash functions. However, no attempt is made here to cover these constructions.

In the following, let ε denote an arbitrary positive constant. Pagh [9] showed that retrieving two entries from the hash table suffices when $r \geq (2 + \varepsilon)n$. Specifically, lookup of a key x can be done by retrieving entries $h_1(x)$ and $h_2(x)$ of the hash table, where h_1 and h_2 are random hash functions mapping machine words to $\{1, \ldots, r\}$. The same result holds if h_1 has range $\{1, \ldots, r/2\}$ and h_2 has range $\{r/2 + 1, \ldots, r\}$, that is, if the two lookups are done in disjoint parts of memory.

It follows from [9] that it is not possible to perform lookup by retrieving a *single* entry in the worst case unless the hash table is of size $n^{2-o(1)}$.

Cuckoo Hashing

Pagh and Rodler [10] showed how to maintain the data structure of Pagh [9] under insertions. They considered the variant in which the lookups are done in disjoint parts of the hash table. It will be convenient to think of these as separate arrays, T_1 and T_2. Let $\perp$ denote the contents of an empty hash table entry, and let $x \leftrightarrow y$ express that the values of variables x and y are swapped. The proposed dynamic algorithm, called *cuckoo hashing*, performs insertions by the following procedure:

```
procedure insert(x)
  i := 1;
  repeat
    x ↔ T_i[h_i(x)]; i := 3 − i;
  until x = ⊥
end
```

At any time the variable x holds a key that needs to be inserted in the table, or $\perp$. The value of i changes between 1 and 2 in each iteration, so the algorithm is alternately exchanging the contents of x with a key from Table 1 and Table 2. Conceptually, what happens is that the algorithm moves a sequence of zero or more keys from one table to the other to make room for the new key. This is done in a greedy fashion, by kicking out any key that may be present in the location where a key is being moved. The similarity of the insertion procedure and the nesting habits of the European cuckoo is the reason for the name of the algorithm.

The pseudocode above is slightly simplified. In general the algorithm needs to make sure not to insert the same key twice, and handle the possibility that the insertion may not succeed (by rehashing if the loop takes too long).

Theorem 1 *Assuming that $r \geq (2 + \varepsilon)n$ the expected time for the cuckoo hashing insertion procedure is $O(1)$.*

Generalizations of Cuckoo Hashing

Cuckoo hashing has been generalized in two directions. First of all, consider the case of k hash functions, for $k > 2$. Second, the hash table may be divided into "buckets" of size b, such that the lookup procedure searches an entire bucket for each hash function. Let (k, b)-cuckoo denote a scheme with k hash functions and buckets of size b. What was described above is a $(2, 1)$-*cuckoo* scheme. Already in 1999, $(4, 1)$-cuckoo was described in a patent application by David A. Brown (US patent 6,775,281). Fotakis et al. described and analyzed a $(k, 1)$-cuckoo scheme in [7], and a $(2, b)$-cuckoo scheme was described and analyzed by Dietzfelbinger and Weidling [4]. In both cases, it was shown that space utilization arbitrarily close to 100% is possible, and that the necessary fraction of unused space decreases exponentially with k and b. The insertion procedure con-

sidered in [4,7] is a breadth first search for the shortest sequence of key moves that can be made to accommodate the new key. Panigrahy [11] studied $(2, 2)$-cuckoo schemes in detail, showing that a space utilization of 83% can be achieved dynamically, still supporting constant time insertions using breadth first search. Independently, Fernholz and Ramachandran [6] and Cain, Sanders, and Wormald [2] determined the highest possible space utilization for $(2, k)$-cuckoo hashing in a *static* setting with no insertions. For $k = 2, 3, 4, 5$ the maximum space utilization is roughly 90%, 96%, 98%, and 99%, respectively.

Applications

Dictionaries have a wide range of uses in computer science and engineering. For example, dictionaries arise in many applications in string algorithms and data structures, database systems, data compression, and various information retrieval applications. No attempt is made to survey these further here.

Open Problems

The results above provide a good understanding of the properties of open addressing schemes with worst case constant lookup time. However, several aspects are still not understood satisfactorily.

First of all, there is no practical class of hash functions for which the above results can be shown. The only explicit classes of hash functions that are known to make the methods work either have evaluation time $\Theta(\log n)$ or use space $n^{\Omega(1)}$. It is an intriguing open problem to construct a class having constant evaluation time and space usage.

For the generalizations of cuckoo hashing the use of breadth first search is not so attractive in practice, due to the associated overhead in storage. A simpler approach that does not require any storage is to perform a random walk where keys are moved to a random, alternative position. (This generalizes the cuckoo hashing insertion procedure, where there is only one alternative position to choose.) Panigrahy [11] showed that this works for $(2, 2)$-cuckoo when the space utilization is low. However, it is unknown whether this approach works well as the space utilization approaches 100%.

Finally, many of the analyzes that have been given are not tight. In contrast, most classical open addressing schemes have been analyzed very precisely. It seems likely that precise analysis of cuckoo hashing and its generalizations is possible using techniques from analysis of algorithms, and tools from the theory of random graphs. In particular, the relationship between space utilization and insertion time is not well understood. A precise analysis of the probability that cuckoo hashing fails has been given by Kutzelnigg [8].

Experimental Results

All experiments on cuckoo hashing and its generalizations so far presented in the literature have been done using simple, heuristic hash functions. Pagh and Rodler [10] presented experiments showing that, for space utilization 1/3, cuckoo hashing is competitive with open addressing schemes that do not give a worst case guarantee. Zukowski et al. [12] showed how to implement cuckoo hashing such that it runs very efficiently on pipelined processors with the capability of processing several instructions in parallel. For hash tables that are small enough to fit in cache, cuckoo hashing was 2 to 4 times faster than chained hashing in their experiments. Erlingsson et al. [5] considered (k, b)-cuckoo schemes for various combinations of small values of k and b, showing that very high space utilization is possible even for modestly small values of k and b. For example, a space utilization of 99.9% is possible for $k = b = 4$. It was further found that the resulting algorithms were very robust. Experiments in [7] indicate that the random walk insertion procedure performs as well as one could hope for.

Cross References

- ► Dictionary Matching and Indexing (Exact and with Errors)
- ► Load Balancing

Recommended Reading

1. Azar, Y., Broder, A.Z., Karlin, A.R., Upfal, E.: Balanced allocations. SIAM J. Comput. **29**(1), 180–200 (1999)
2. Cain, J.A., Sanders, P., Wormald, N.: The random graph threshold for k-orientability and a fast algorithm for optimal multiple-choice allocation. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07), pp. 469–476. ACM Press, New Orleans, Louisiana, USA, 7–9 December 2007
3. Carter, J.L., Wegman, M.N.: Universal classes of hash functions. J. Comput. Syst. Sci. **18**(2), 143–154 (1979)
4. Dietzfelbinger, M., Weidling, C.: Balanced allocation and dictionaries with tightly packed constant size bins. In: ICALP. Lecture Notes in Computer Science, vol. 3580, pp. 166–178. Springer, Berlin (2005)
5. Erlingsson, Ú., Manasse, M., McSherry, F.: A cool and practical alternative to traditional hash tables. In: Proceedings of the 7th Workshop on Distributed Data and Structures (WDAS '06), Santa Clara, CA, USA, 4–6 January 2006
6. Fernholz, D., Ramachandran, V.: The k-orientability thresholds for $g_{n,p}$. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07), pp. 459–468. ACM Press, New Orleans, Louisiana, USA, 7–9 December 2007

7. Fotakis, D., Pagh, R., Sanders, P., Spirakis, P.G.: Space efficient hash tables with worst case constant access time. Theor. Comput. Syst. **38**(2), 229–248 (2005)
8. Kutzelnigg, R.: Bipartite Random Graphs and Cuckoo Hashing. In: Proc. Fourth Colloquium on Mathematics and Computer Science, Nancy, France, 18–22 September 2006
9. Pagh, R.: On the cell probe complexity of membership and perfect hashing. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC '01), pp. 425–432. ACM Press, New York (2001)
10. Pagh, R., Rodler, F.F.: Cuckoo hashing. J. Algorithms **51**, 122–144 (2004)
11. Panigrahy, R.: Efficient hashing with lookups in two memory accesses. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '05), pp. 830–839. SIAM, Vancouver, 23–25 January 2005
12. Zukowski, M., Heman, S., Boncz, P.A.: Architecture-conscious hashing. In: Proceedings of the International Workshop on Data Management on New Hardware (DaMoN), Article No. 6. ACM Press, Chicago, Illinois, USA, 25 June 2006