W

Wait-Free Consensus

► Asynchronous Consensus Impossibility

Wait-Free Registers

► Registers

Wait-Free Renaming

► Renaming
► Topology Approach in Distributed Computing

Wait-Free Shared Variables

► Registers

Wait-Free Synchronization

1991; Herlihy

MARK MOIR
Sun Microsystems Laboratories, Burlington, MA, USA

Problem Definition

The traditional use of locking to maintain consistency of shared data in concurrent programs has a number of disadvantages related to software engineering, robustness, performance, and scalability. As a result, a great deal of research effort has gone into *nonblocking* synchronization mechanisms over the last few decades.

Herlihy's seminal paper *Wait-Free Synchronization* [12] studied the problem of implementing concurrent data structures in a *wait-free manner*, i. e., so that every operation on the data structure completes in a finite number of steps by the invoking thread, regardless of how fast or slow other threads run and even if some or all of them halt permanently. Implementations based on locks are not wait-free because, while one thread holds a lock, others can take an unbounded number of steps waiting to acquire the lock. Thus, by requiring implementations to be wait-free, some of the disadvantages of locks may potentially be eliminated.

The first part of Herlihy's paper examined the power of different synchronization primitives for wait-free computation. He defined the *consensus number* of a given primitive as the maximum number of threads for which we can solve wait-free consensus using that primitive (together with read-write registers). The consensus problem requires participating threads to *agree* on a value (e. g., true or false) amongst values proposed by the threads. The ability to solve this problem is a key indicator of the power of synchronization primitives because it is central to many natural problems in concurrent computing. For example, in a software transactional memory system, threads must agree that a particular transaction either committed or aborted.

Herlihy established a *hierarchy* of synchronization primitives according to their consensus number. He showed (i) that the consensus number of read-write registers is 1 (so wait-free consensus cannot be solved for even two threads), (ii) that the consensus number of stacks and FIFO queues is 2, and (iii) that there are so-called *universal* primitives, which have consensus number ∞. Common examples include `compare-and-swap` (`CAS`) and the `load-linked/store-conditional` (`LL/SC`) pair.

There are a number of papers which examine Herlihy's hierarchy in more detail. These show that seemingly minor variations in the model or in the semantics of primitives can have a surprising effect on results. Most of this work is primarily of theoretical interest. The key practical point to take away from Herlihy's hierarchy is that we need universal primitives to support effective wait-free synchronization in general. Recognizing this fact, all modern shared-memory multiprocessors provide some form of universal primitive.

Herlihy additionally showed that a solution to consensus can be used to implement any shared object in a wait-free manner, and thus that any universal primitive suffices for this purpose. He demonstrated this idea using a so-called *universal construction*, which takes sequential code for an object and creates a wait-free implementation of the object using consensus to resolve races between concurrent operations. Despite the important practical ramifications of this result, the universal construction itself was quite impractical. The basic idea was to build a list of operations, using consensus to determine the order of operations, and to allow threads to iterate over the list applying the operations in order to determine the current state of the object. The construction required $O(N^3)$ space to ensure enough operations are retained to allow the current state to be determined. It was also very slow, requiring many threads to recompute the same information, and thus preventing parallelism between operations in addition.

Later, Herlihy [13] presented a more concrete universal construction based on the `LL`/`SC` instruction pair. This construction required $N + 1$ copies of the object for N threads and still did not admit any parallelism; thus it was also not practical. Despite this, work following on from Herlihy's has brought us to the point today that we can support practical programming models that provide nonblocking implementations of arbitrary shared objects. The remainder of this chapter discusses the state of nonblocking synchronization today, and mentions some history along the way.

Weaker Nonblocking Progress Conditions

Various researchers, including us, have had some success attempting to overcome the disadvantages of Herlihy's wait-free constructions. However, the results remain impractical due to excessive overhead and overly complicated algorithms. In fact, there are still no nontrivial wait-free shared objects in widespread practical use, either implemented directly or using universal constructions.

The biggest advances towards practicality have come from considering weaker progress conditions. While theoreticians worked on wait-free implementations, more pragmatic researchers sought lock-free implementations of shared objects. A *lock-free* implementation guarantees that, after a finite number of steps of any operation, *some* operation completes. In contrast to wait-free algorithms, it is in principle possible for one operation of a lock-free data structure to be continually starved by others. However, this rarely occurs in practice, especially because contention control techniques such as exponential backoff [1] are often used to reduce contention when it occurs, which makes repeated interference even more unlikely. Thus, the lack of a strong progress guarantee like wait-freedom has often been found to be acceptable in practice.

The observation that weaker nonblocking progress conditions allow simpler and more practical algorithms led Herlihy et al. [15] to define an even weaker condition: An *obstruction-free* algorithm does not guarantee that an operation completes unless it eventually encounters no more interference from other operations. In our experience, obstruction-free algorithms are easier to design, simpler, and faster in the common uncontended case than lock-free algorithms. The price paid for these benefits is that obstruction-free algorithms can "livelock", with two or more operations repeatedly interfering with each other forever. This is not merely a theoretical concern: it has been observed to occur in practice [16]. Fortunately, it is usually straightforward to eliminate livelock in practice through contention control mechanisms that control and manipulate when operations are executed to avoid repeated interference.

The obstruction-free approach to synchronization is thus to design simple and efficient algorithms for the weak obstruction-free progress condition, and to integrate orthogonal contention control mechanisms to facilitate progress when necessary. By largely separating the difficult issues of correctness and progress, we significantly ease the task of designing effective nonblocking implementations: the algorithms are not complicated by tightly coupled mechanisms for achieving lock-freedom, and it is easy to modify and experiment with contention control mechanisms because they are separate from the algorithm and do not affect its correctness. We have found this approach to be very powerful.

Transactional Memory

The severe difficulty of designing and verifying correct nonblocking data structures has led researchers to investigate the use of tools to produce them, rather than designing them directly. In particular, *transactional memory* ([5,17,23]) has emerged as a promising direction. Transactional memory allows programmers to express sections of code that should be executed atomically, and the transactional memory system (implemented in hardware, software, or a combination of the two) is responsible for managing interactions between concurrent transactions to ensure this atomicity. Here we concentrate on software transactional memory (STM).

The progress guarantee made by a concurrent data structure implemented using STM depends on the STM

implementation. It is possible to characterize the progress conditions of transactional memory implementations in terms of a system of threads in which each operation on a shared data structure is executed by repeatedly attempting to apply it using a transaction until an attempt successfully commits. In this context, say the transactional memory implementation is obstruction-free if it guarantees that, if a thread repeatedly executes transactions and eventually encounters no more interference from other threads, then it eventually successfully commits a transaction.

Key Results

This section briefly discusses some of the most relevant results concerning nonblocking synchronization, and obstruction-free synchronization in particular.

While progress towards practicality was made with lock-free implementations of shared objects as well as lock-free STM systems, this progress was slow because simultaneously ensuring correctness and lock-freedom proved difficult. Before the introduction of obstruction-freedom, the lock-free STMs still had some severe disadvantages such as the need to declare and initialize all memory to be accessed by transactions in advance, the need for transactions to know in advance which memory locations they will access, unacceptable constraints on the layout of such memory, etc.

In addition to the work on tools such as STM for building nonblocking data structures, there has been a considerable amount of work on direct implementations. While this work has not yielded any practical wait-free algorithms, a handful of practical lock-free implementations for simple data structures such as queues and stacks have been achieved [21,24]. There are also a few slightly more ambitious implementations in the literature that are arguably practical, but the algorithms are complicated and subtle, many are incorrect, and almost none has a formal proof. Proofs for such algorithms are challenging, and minor changes to the algorithm require the proofs to be redone.

The next section, discusses some of the results that have been achieved by applying the obstruction-free approach. The remainder of this section, briefly discusses a few results related to the approach itself.

An important practical aspect of using an obstruction-free algorithm is how contention is managed when it arises. In introducing obstruction-freedom, Herlihy et al. [15] explained that contention control is necessary to facilitate progress in the face of contention because obstruction-free algorithms do not directly make any progress guarantee in this case. However, they did not directly address *how* contention control mechanisms could be used in practice.

Subsequently, Herlihy et al. [16] presented a dynamic STM system (see next section) that provides an interface for a modular contention manager, allowing for experimentation with alternative contention managers. Scherer and Scott [22] experimented with a number of alternatives, and found that the best contention manager depends on the workload. Guerraoui et al. [9] described an implementation that supports changing contention managers on the fly in response to changing workload conditions.

All of the contention managers discussed in the abovementioned papers are ad hoc contention managers based on intuition; no analysis is given of what guarantees (if any) are made by the contention managers. Guerraoui et al. [10] made a first step towards a formal analysis of contention managers by showing that their `Greedy` contention manager guarantees that every transaction eventually completes. However, using the `Greedy` contention manager results in a blocking algorithm, so their proof necessarily assumes that threads do not fail while executing transactions.

Fich et al. [7] showed that any obstruction-free algorithm can be automatically transformed into one that is *practically* wait-free in any real system. "Practically" is said because the wait-free progress guarantee depends on partial synchrony that exists in any real system, but the transformed algorithm is not technically wait-free, because this term is defined in the context of a fully asynchronous system. Nonetheless, an algorithm achieved by applying the transformation of Fich et al. to an obstruction-free algorithm does guarantee progress to non-failed transactions, even if other transactions fail.

Work on incorporating contention management techniques into obstruction-free algorithms has mostly been done in the context of STM, so the contention manager can be called directly from the STM implementation. Thus, the programmer using the STM need not be concerned with how contention management is integrated, but this does not address how contention management is integrated into direct implementations of obstruction-free data structures.

One option is for the programmer to manually insert calls to a contention manager, but this approach is tedious and error prone. Guerraoui et al. [11] suggested a version of this approach in which the contention manager is abstracted out as a failure detector. They also explored what progress guarantees can be made by what failure detectors.

Attiya et al. [4] and Aguilera et al. [2] suggested changing the semantics of the data structure's operations so

that they can return a special value in case of contention, thus allowing contention management to be done outside the data structure implementation. These approaches still leave a burden on the programmer to ensure that these special values are always returned by an operation that cannot complete due to contention, and that the correct special value is returned according to the prescribed semantics.

Another option is to use system support to ensure that contention management calls are made frequently enough to ensure progress. This support could be in the form of compiled-in calls, runtime support, signals sent upon expiration of a timer, etc. But all of these approaches have disadvantages such as not being applicable in general purpose environments, not being portable, etc.

Given that it remains challenging to design and verify direct obstruction-free implementations of shared data structures, and that there are disadvantages to the various proposals for integrating contention control mechanisms into them, using tools such as STMs with built-in contention management interfaces is the most convenient way to build nonblocking data structures.

Applications

The obstruction-free approach to nonblocking synchronization was introduced by Herlihy et al. [15], who used it to design a double-ended queue (deque) based on the widely available `CAS` instruction. All previous nonblocking deques either require exotic synchronization instructions such as `double-compare-and-swap` (`DCAS`), or have the disadvantage that operations at opposite ends of the queue always interfere with each other.

Herlihy et al. [16] introduced Dynamic STM (DSTM), the first STM that is dynamic in the following two senses: new objects can be allocated on the fly and subsequently accessed by transactions, and transactions do not need to know in advance what objects will be accessed. These two advantages made DSTM much more useful than previous STMs for programming dynamic data structures. As a result, nonblocking implementations of sophisticated shared data structures such as balanced search trees, skip lists, dynamic hash tables, etc. were suddenly possible.

The obstruction-free approach played a key role in the development of both of the results mentioned above: Herlihy et al. [16] could concentrate on the functionality and correctness of DSTM without worrying about how to achieve stronger progress guarantees such as lock-freedom.

The introduction of DSTM and of the obstruction-free approach have led to numerous improvements and variations by a number of research groups, and most of these have similarly followed the obstruction-free approach. However, Harris and Fraser [8] presented a dynamic STM called OSTM with similar advantages to DSTM, but it is lock-free. Experiments conducted at the University of Rochester [20] showed that DSTM outperformed OSTM by an order of magnitude on some workloads, but that OSTM outperformed DSTM by a factor of 2 on others. These differences are probably due to various design decisions that are (mostly) orthogonal to the progress condition, so it is not clear what we can conclude about how the choice of progress condition affects performance in this case.

Perhaps a more direct comparison can be made between another pair of algorithms, again an obstruction-free one by Herlihy et al. [14] and a similar but lock-free one by Harris and Fraser [8]. These algorithms, invented independently of each other, implement `MCAS` (`CAS` generalized to access M independently chosen memory locations). The two algorithms are very similar, and a close comparison revealed that the only real differences between them were due to Harris and Fraser's desire to have a lock-free implementation. As a result of this, their algorithm is somewhat more complicated, and also requires a minimum of $3M + 1$ `CAS` operations, whereas the algorithm of Herlihy et al. [14] requires only $2M + 1$. The authors are unaware of any direct performance comparison of these algorithms, but they believe the obstruction-free one would outperform the lock-free one, particularly in the absence of conflicting `MCAS` operations.

Open Questions

Because transactional memory research has grown out of research into nonblocking data structures, it was long considered mandatory for STM implementations to support the development of nonblocking data structures. Recently, however, a number of researchers have observed that at least the software engineering benefits of transactional memory can be delivered even by a blocking STM. There are ongoing debates whether STM needs to be nonblocking and whether there is a fundamental cost to being nonblocking.

While we agree that blocking STMs are considerably easier to design, and that in many cases a blocking STM is acceptable, this is not always true. Consider, for example, an interrupt handler that shares data with the interrupted thread. The interrupted thread will not run again until the interrupt handler completes, so it is critical that the interrupted thread does not block the interrupt handler. Thus, if using STM is desired to simplify the code for accessing

this shared data, the STM *must* be nonblocking. The authors are therefore motivated to continue research aimed at improving nonblocking STMs and to understand what fundamental gap, if any, exists between blocking and nonblocking STMs.

Progress in improving the common-case performance of nonblocking STMs continues [19], and the authors see no reason to believe that nonblocking STMs should not be very competitive with blocking STMs in the common case, i. e., until the system decides that one transaction should not wait for another that is delayed (an option that is not available with blocking STMs).

It is conjectured that indeed a separation between blocking and nonblocking STMs can be proved according to some measure, but that this will not imply significant performance differences in the common case. Indeed results of Attiya et al. [3] show a separation between obstruction-free and blocking algorithms according to a measure that counts the number of distinct base objects accessed by the implementation plus the number of "memory stalls", which measure how often the implementation can encounter contention for a variable from another thread. While this result is interesting, it is not clear that it is useful for deciding whether to implement blocking or obstruction-free objects, because the measure does not account for the time spent waiting by blocking implementations, and thus is biased in their favor. For now, remain optimistic that STMs can be made to be nonblocking without paying a severe performance price in the common case.

Another interesting question, which is open as far as the authors know, is whether there is a fundamental cost to implementing stronger nonblocking progress conditions versus obstruction-freedom. Again, they conjecture that there is. It is known that there is a fundamental difference between obstruction-freedom and lock-freedom in systems that support only reads and writes: It is possible to solve obstruction-free consensus but not lock-free consensus in this model [15]. While this is a fascinating observation, it is mostly irrelevant from a practical standpoint as all modern shared memory multiprocessors support stronger synchronization primitives such as CAS, with which it is easy to solve consensus, even wait-free. The interesting question therefore is whether there is a fundamental cost to being lock-free as opposed to obstruction-free in real systems.

To have a real impact on design directions, such results need to address common case performance, or some other measure (perhaps space) that is relevant to everyday use. Many lower bound results establish a separation in worst-case time complexity, which does not necessarily have a direct impact on design decisions, because the worst case may be very rare. So far, efforts to establish a separation according to potentially useful measures have only led to stronger results than we had conjectured were possible. In the authors first attempt [18], they tried to establish a separation in the number of CAS instructions needed in the absence of contention to solve consensus, but found that this was not a very useful measure, as were able to come up with a wait-free implementation that avoids CAS in the absence of contention. The second attempt [6] was to establish a separation according to the *obstruction-free step complexity* measure, which counts the maximum number of steps to complete an operation once the operation encounters no more contention. They knew we could implement obstruction-free DCAS with constant obstruction-free step complexity, and attempt to prove this impossible for lock-free DCAS, but achieved such an algorithm. These experiences suggest that, in addition to their direct advantages, obstruction-free algorithms may provide a useful stepping stone to algorithms with stronger progress properties.

Finally, while a number of contention managers have proved effective for various workloads, it is an open question whether a single contention manager can adapt to be competitive with the best on all workloads, and how close it can come to making optimal contention management decisions. Experience to date suggests that this will be very challenging to achieve. Therefore, as in any system, the first priority should be avoiding contention in the first place. Fortunately, transactional memory has the potential to make this much easier than in lock-based programming models, because it offers the benefits of fine-grained synchronization without the programming complexity that accompanies fine-grained locking schemes.

Cross References

► Concurrent Programming, Mutual Exclusion
► Linearizability

Recommended Reading

1. Agarwal, A., Cherian, M.: Adaptive backoff synchronization techniques. In: Proceedings of the 16th Annual International Symposium on Computer Architecture, pp. 396–406. ACM Press, New York (1989)
2. Aguilera, M.K., Frolund, S., Hadzilacos, V., Horn, S.L., Toueg, S.: Brief announcement: Abortable and query-abortable objects. In: Proc. 20th Annual International Symposium on Distributed Computing, 2006
3. Attiya, H., Guerraoui, R., Hendler, D., Kouznetsov, P.: Synchronizing without locks is inherently expensive. In: PODC '06: Proceedings of the twenty-fifth Annual ACM Symposium on Prin-

ciples of Distributed Computing, New York, USA, pp. 300–307. ACM Press (2006)
4. Attiya, H., Guerraoui, R., Kouznetsov, P.: Computing with reads and writes in the absence of step contention. In: Proc. 19th Annual International Symposium on Distributed Computing, 2005
5. Damron, P., Fedorova, A., Lev, Y., Luchangco, V., Moir, M., Nussbaum, D.: Hybrid transactional memory. In: Proc. 12th Symposium on Architectural Support for Programming Languages and Operating Systems, 2006
6. Fich, F., Luchangco, V., Moir, M., Shavit, N.: Brief announcement: Obstruction-free step complexity: Lock-free DCAS as an example. In: Proc. 19th Annual International Symposium on Distributed Computing, 2005
7. Fich, F., Luchangco, V., Moir, M., Shavit, N.: Obstruction-free algorithms can be practically wait-free. In: Proc. 19th Annual International Symposium on Distributed Computing, 2005
8. Fraser, K., Harris, T.: Concurrent programming without locks. http://www.cl.cam.ac.uk/netos/papers/2004-cpwl-submission.pdf (2004)
9. Guerraoui, R., Herlihy, M., Pochon, B.: Polymorphic contention management. In: Proc. 19th Annual International Symposium on Distributed Computing, 2005
10. Guerraoui, R., Herlihy, M., Pochon, B.: Toward a theory of transactional contention managers. In: Proc. 24th Annual ACM Symposium on Principles of Distributed Computing, 2005, pp. 258–264
11. Guerraoui, R., Kapalka, M., Kouznetsov, P.: The weakest failure detector to boost obstruction freedom. In: Proc. 20th Annual International Symposium on Distributed Computing, 2006
12. Herlihy, M.: Wait-free synchronization. ACM Trans. Program. Lang. Syst. **13**(1), 124–149 (1991)
13. Herlihy, M.: A methodology for implementing highly concurrent data objects. ACM Trans. Program. Lang. Syst. **15**(5), 745–770 (1993)
14. Herlihy, M., Luchangco, V., Moir, M.: Obstruction-free mechanism for atomic update of multiple non-contiguous locations in shared memory. US Patent Application 20040034673 (2002)
15. Herlihy, M., Luchangco, V., Moir, M.: Obstruction-free synchronization: Double-ended queues as an example. In: Proceedings of the 23rd International Conference on Distributed Computing Systems, 2003
16. Herlihy, M., Luchangco, V., Moir, M., Scherer III., W.: Software transactional memory for supporting dynamic-sized data structures. In: Proc. 22th Annual ACM Symposium on Principles of Distributed Computing, 2003, pp. 92–101
17. Herlihy, M., Moss, J.E.B.: Transactional memory: Architectural support for lock-free data structures. In: Proc. 20th Annual International Symposium on Computer Architecture, 1993, pp. 289–300
18. Luchangco, V., Moir, M., Shavit, N.: On the uncontended complexity of consensus. In: Proc. 17th Annual International Symposium on Distributed Computing, 2005
19. Marathe, V.J., Moir, M.: Toward high performance nonblocking software transactional memory. In: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. pp. 227–236, ACM, New York, USA (2008)
20. Marathe, V., Scherer, W., Scott, M.: Adaptive software transactional memory. In: Proc. 19th Annual International Symposium on Distributed Computing, 2005
21. Michael, M., Scott, M.: Nonblocking algorithms and preemption-safe locking on multiprogrammed shared memory multiprocessors. J. Parall. Distrib. Comput. **51**(1), 1–26 (1998)
22. Scherer, W., Scott, M.: Advanced contention management for dynamic software transactional memory. In: Proc. 24th Annual ACM Symposium on Principles of Distributed Computing, 2005
23. Shavit, N., Touitou, D.: Software transactional memory. Distrib. Comput., Special Issue **10**, 99–116 (1997)
24. Treiber, R.: Systems programming: Coping with parallelism. Technical Report RJ5118, IBM Almaden Research Center (1986)

Warehouse Location

► Facility Location
► Local Search for *K*-medians and Facility Location

Weighted Bipartite Matching

► Assignment Problem

Weighted Caching

► Online Paging and Caching

Weighted Connected Dominating Set

2005; Wang, Wang, Li

Yu Wang[1], Weizhao Wang[2], Xiang-Yang Li[3]
[1] Department of Computer Science, University of North Carolina at Charlotte, Charlotte, NC, USA
[2] Google Inc., Irvine, CA, USA
[3] Department of Computer Science, Illinois Institue of Technology, Chicago, IL, USA

Keywords and Synonyms

Minimum weighted connected dominating set

Problem Definition

This problem is concerned with a weighted version of the classical minimum connected dominating set problem. This problem has numerous motivations including wireless networks and distributed systems. Previous work [1,2,4,5,6,14] in wireless networks focuses on designing efficient distributed algorithms to construct the connected dominating set which can be used as the virtual

backbone for the network. Most of the proposed methods try to minimize the number of nodes in the backbone (i. e., the number of clusterheads). However, in many applications, minimizing the size of the backbone is not sufficient. For example, in wireless networks different wireless nodes may have different costs for serving as a clusterhead, due to device differences, power capacities, and information loads to be processed. Thus, by assuming each node has a cost to being in the backbone, there is a need to study distributed algorithms for weighted backbone formation. Centralized algorithms to construct a weighted connected dominating set with minimum weight have been studied [3,7,9]. Recently, the work of Wang, Wang, and Li [12,13] proposes an efficient distributed method to construct a weighted backbone with low cost. They proved that the total cost of the constructed backbone is within a small constant factor of the optimum when either the nodes' costs are smooth (i. e. the maximum ratio of costs of adjacent nodes is bounded) or the network maximum node degree is bounded. To the best knowledge of the entry authors, this work is the first to consider this weighted version of minimum connected dominating set problem and provide a distributed approximation algorithm.

Notations

A communication graph $G = (V, E)$ over a set V of wireless nodes has an edge uv between nodes u and v if and only if u and v can communicate directly with each other, i. e., inside the transmission region of each other. Let $d_G(u)$ be the degree of node u in a graph G and Δ be the maximum node degree of all wireless nodes (i. e. $\Delta = \max_{u \in V} d_G(u)$). Each wireless node u has a cost $c(u)$ of being in the backbone. Let $\delta = \max_{ij \in E} c(i)/c(j)$, where ij is the edge between nodes i and j, E is the set of communication links in the wireless network G, and the maximum operation is taken on all pairs of adjacent nodes i and j in G. In other words, δ is the maximum ratio of costs of two adjacent nodes and can be called the *cost smoothness* of the network. When δ is bounded by some small constant, the node costs are *smooth*. When the transmission region of every wireless node is modeled by a unit disk centered at itself, the communication graph is often called a *unit disk graph*, denoted by $UDG(V)$. Such networks are also called *homogeneous networks*.

A subset S of V is a *dominating set* if each node in V is either in S or is adjacent to some node in S. Nodes from S are called dominators, while nodes not in S are called dominatees. A subset B of V is a *connected dominating set* (CDS) if B is a dominating set and B induces a connected subgraph. Consequently, the nodes in B can communicate with each other without using nodes in $V - B$. A dominating set with minimum cardinality is called *minimum dominating set* (MDS). A CDS with minimum cardinality is the *minimum connected dominating set* (MCDS). In the weighted version, assume that each node u has a cost $c(u)$. Then a CDS B is called *weighted connected dominating set* (WCDS). A subset B of V is a *minimum weighted connected dominating set* (MWCDS) if B is a WCDS with minimum total cost. It is well-known that finding either the *minimum connected dominating set* or the *minimum weighted connected dominating set* is a NP-hard problem even when G is a unit disk graph. The work of Wang et al. studies efficient approximation algorithms to construct a low-cost backbone which can approximate the MWCDS problem well. For a given communication graph $G = (V, E, C)$ where V is the set of nodes, E is the edge set, and C is the set of weights for edges, the corresponding minimum weighted connected dominating set problem is as follows.

Problem 1 (Minimum Weighted Connected Dominating Set)
INPUT: *The weighted communication graph* $G = (V, E, C)$.
OUTPUT: *A subset* A *of* V *is a minimum weighted connected dominating set, i. e.,* (1) *A is a dominating set;* (2) *A induces a connected subgraph;* (3) *the total cost of A is minimum.*

Another related problem is independent set problem. A subset of nodes in a graph G is an *independent set* if for any pair of nodes, there is no edge between them. It is a *maximal independent set* if no more nodes can be added to it to generate a larger independent set. Clearly, any maximal independent set is a dominating set. It is a *maximum independent set* (MIS) if no other independent set has more nodes. The independence number, denoted as $\alpha(G)$, of a graph G is the size of the MIS of G. The *k-local independence number*, denoted by $\alpha^{[k]}(G)$, is defined as $\alpha^{[k]}(G) = \max_{u \in V} \alpha(G_k(u))$. Here, $G_k(u)$ is the induced graph of G on k-hop neighbors of u (denoted by $N_k(u)$), i. e., $G_k(u)$ is defined on $N_k(u)$, and contains all edges in G with both end-points in $N_k(u)$. It is well-known that for a unit disk graph, $\alpha^{[1]}(UDG) \leq 5$ [2] and $\alpha^{[2]}(UDG) \leq 18$ [11].

Key Results

Since finding the minimum weighted connected dominating set (MWCDS) is NP-hard, centralized approximation algorithms for MWCDS have been studied [3,7,9]. In [9], Klein and Ravi proposed an approximation algorithm for the node-weighted Steiner tree problem. Their algorithm

can be generalized to compute a $O(\log \Delta)$ approximation for MWCDS. Guha and Khuller [7] also studied the approximation algorithms for node-weighted Steiner tree problem and MWCDS. They developed an algorithm for MWCDS with an approximation factor of $(1.35 + \epsilon) \log \Delta$ for any fixed $\epsilon > 0$. Recently, Ambuhl et al. [3] provided a constant approximation algorithm for MWCDS under UDG model. Their approximation ratio is bounded by 89. All these algorithms are centralized algorithms, while the applications in wireless ad hoc networks prefer distributed solutions for MWCDS.

In [12,13], Wang et al. proposed a distributed algorithm that constructs a weighted connected dominating set for a wireless ad hoc network G. Their method has two phases: the first phase (clustering phase, Algorithm 1 in [12,13]) is to find a set of wireless nodes as the dominators (clusterheads) and the second phase (Algorithm 2 in [12,13]) is to find a set of nodes, called *connectors*, to connect these dominators to form the final backbone. Wang et al. proved that the total cost of the constructed backbone is no more than $\min(\alpha^{[2]}(G)\log(\Delta+1), (\alpha^{[1]}(G)-1)\delta+1) + 2\alpha^{[1]}(G)$ times of the optimum solution.

Algorithm 1 first constructs a maximal independent set (MIS) using classical greedy method with the node cost as the selection criterion. For each node v in MIS, it then runs a local greedy set cover method on the *local neighborhood* $N_2(v)$ to find some nodes ($GRDY_v$) to cover all one-hop neighbors of v. If $GRDY_v$ has a total cost smaller than v, then it uses $GRDY_v$ to replace v, which further reduces the cost of MIS. The following theorem of the total cost of this selected set is proved in [12,13].

Theorem 1 *For a network modeled by a graph G, Algorithm 1 (in [12,13]) constructs a dominating set whose total cost is no more than* $\min(\alpha^{[2]}(G)\log(\Delta+1), (\alpha^{[1]}(G)-1)\delta+1)$ *times of the optimum.*

Algorithm 2 finds some *connectors* among all the dominatees to connect the dominators into a backbone (CDS). It forms a CDS by finding connectors to connect any pair of dominators u and v if they are connected in the original graph G with at most 3 hops. A distributed algorithm to build a MST then is performed on the CDS. The following theorem of the total cost of these connectors is proved in [12,13].

Theorem 2 *The connectors selected by Algorithm 2 (in [12,13]) have a total cost no more than* $2 \cdot \alpha^{[1]}(G)$ *times of the optimum for networks modeled by G.*

Combining Theorem 1 and Theorem 2, the following theorem is the main contributions of the work of Wang et al..

Theorem 3 *For any communication graph G, Algorithm 1 and Algorithm 2 construct a weighted connected dominating set whose total cost is no more than*

$$\min(\alpha^{[2]}(G)\log(\Delta+1), (\alpha^{[1]}(G)-1)\delta+1) + 2\alpha^{[1]}(G)$$

times of the optimum.

Notice that, for homogeneous wireless networks modeled by UDG, it implies that the constructed backbone has a cost no more than $\min(18\log(\Delta+1), 4\delta+1)+10$ times of the optimum. The advantage of the constructed backbone is that the total cost is small compared with the optimum when either the costs of wireless nodes are smooth, *i. e.*, two neighboring nodes' costs differ by a small constant factor, or the maximum node degree is low.

In term of time complexity, the most time-consuming step in the proposed distributed algorithm is building the MST. In [10], Kuhn et al. gave a lower bound on the distributed time complexity of any distributed algorithm that wants to compute a minimum dominating set in a graph. Essentially, they proved that even for the unconnected and unweighted case, any distributed approximation algorithm with poly-logarithmic approximation guarantee for the problem has to have a time-complexity of at least $\Omega(\log \Delta / \log\log \Delta)$.

Applications

The proposed distributed algorithms for MWCDS can be used in ad hoc networks or distributed system to form a low-cost network backbone for communication application. The cost used as the input of the algorithms could be a *generic* cost, defined by various practical applications. It may represent the *fitness* or *priority* of each node to be a clusterhead. The lower cost means the higher priority. In practice, the cost could represent the power consumption rate of the node if a backbone with small power consumption is needed; the robustness of the node if fault-tolerant backbone is needed; or a function of its security level if a secure backbone is needed; or a combined weight function to integrate various metrics such as traffic load, signal overhead, battery level, and coverage. Therefore, by defining different costs, the proposed low-cost backbone formation algorithms can be used in various practical applications. Beside forming the backbone for routing, the weighted clustering algorithm (Algorithm 1) can also be used in other applications, such as selecting the mobile agents to perform intrusion detection in ad hoc networks [8] (to achieve more robust and power efficient agent selection), or select the rendezvous points to collect and store data in sensor networks [15] (to achieve the energy efficiency and storage balancing).

Open Problems

A number of problems related to the work of Wang, Wang, and Li [12,13] remain open. The proposed method assumes that the nodes are almost-static in a reasonable period of time. However, in some network applications, the network could be highly dynamic (both the topology or the cost could change). Therefore, after the generation of the weighted backbone, the dynamic maintenance of the backbone is also an important issue. It is still unknown how to update the topology efficiently while preserving the approximation quality.

In [12,13], the following assumptions on wireless network model is used: omni-directional antenna, single transmission received by all nodes within the vicinity of the transmitter. The MWCDS problem will become much more complicated if some of these assumptions are relaxed.

Experimental Results

In [12,13], simulations on random networks are conducted to evaluate the performances of the proposed weighted backbone and several backbones built by previous methods. The simulation results confirm the theoretical results.

Cross References

► Connected Dominating Set

Recommended Reading

1. Alzoubi, K., Wan, P.-J., Frieder, O.: New distributed algorithm for connected dominating set in wireless ad hoc networks. In: Proceedings of IEEE 35th Hawaii International Conference on System Sciences (HICSS-35), Hawaii, 7–10 January 2002
2. Alzoubi, K., Li, X.-Y., Wang, Y., Wan, P.-J., Frieder, O.: Geometric spanners for wireless ad hoc networks. IEEE Trans. Parallel Distrib. Process. **14**, 408–421 (2003)
3. Ambuhl, C., Erlebach, T., Mihalak, M., Nunkesser, M.: Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In: Proceedings of the 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2006), Barcelona, 28–30 August 2006, LNCS, vol. 4110, pp. 3–14. Springer, Berlin Heidelberg (2006)
4. Bao, L., Garcia—Aceves, J.J.: Topology management in ad hoc networks. In: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing, Annapolis, 1–3 June 2003, pp. 129–140. ACM Press, New York (2003)
5. Chatterjee, M., Das, S., Turgut, D.: WCA: A weighted clustering algorithm for mobile ad hoc networks. J. Clust. Comput. **5**, 193–204 (2002)
6. Das, B., Bharghavan, V.: Routing in ad-hoc networks using minimum connected dominating sets. In: Proceedings of IEEE International Conference on on Communications (ICC'97), vol. 1, pp. 376–380. Montreal, 8–12 June 1997
7. Guhaa, S., Khuller, S.: Improved methods for approximating node weighted Steiner trees and connected dominating sets. Inf. Comput. **150**, 57–74 (1999)
8. Kachirski, O., Guha, R.: Intrusion detection using mobile agents in wireless ad hoc networks. In: Proceedings of IEEE Workshop on Knowledge Media Networking, Kyoto, 10–12 July 2002
9. Klein, P., Ravi, R.: A nearly best-possible approximation algorithm for node-weighted Steiner trees. J. Algorithms **19**, 104–115 (1995)
10. Kuhn, F., Moscibroda, T., Wattenhofer, R.: What cannot be computed locally! In: Proceedings of the 23rd ACM Symposium on the Principles of Distributed Computing (PODC), St. John's, July (2004)
11. Li, X.-Y., Wan, P.-J.: Theoretically good distributed CDMA/OVSF code assignment for wireless ad hoc networks. In: Proceedings of 11th Internatioanl Computing and Combinatorics Conference (COCOON), Kunming, 16–19 August 2005
12. Wang, Y., Wang, W., Li, X.-Y.: Efficient distributed low-cost backbone formation for wireless networks. In: Proceedings of 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2005), Urbana-Champaign, 25–27 May 2005
13. Wang, Y., Wang, W., Li, X.-Y.: Efficient distributed low cost backbone formation for wireless networks. IEEE Trans. Parallel Distrib. Syst. **17**, 681–693 (2006)
14. Wu, J., Li, H.: A dominating-set-based routing scheme in ad hoc wireless networks. The special issue on Wirel. Netw. Telecommun. Systems J. **3**, 63–84 (2001)
15. Zheng, R., He, G., Gupta, I., Sha, L.: Time idexing in sensor networks. In: Proceedings of 1st IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS), Fort Lauderdale, 24–27 October 2004

Weighted Popular Matchings

2006; Mestre

Julián Mestre
Department of Computer Science, University of Maryland, College Park, MD, USA

Problem Definition

Consider the problem of matching a set of individuals X to a set of items Y where each individual has a weight and a personal preference over the items. The objective is to construct a matching M that is stable in the sense that there is no matching M' such that the weighted majority vote will choose M' over M.

More formally, a bipartite graph (X, Y, E), a weight $w(x) \in R^+$ for each individual $x \in X$, and a rank function $r : E \to \{1, \ldots, |Y|\}$ encoding the individual preferences are given. For every applicant x and items $y_1, y_2 \in Y$ say applicant x prefers y_1 over y_2 if $r(x, y_1) < r(x, y_2)$, and x is indifferent between y_1 and y_2 if $r(x, y_1) = r(x, y_2)$.

The preference lists are said to be strictly ordered if applicants are never indifferent between two items, otherwise the preference lists are said to contain ties.

Let M and M' be two matchings. An applicant x prefers M over M' if x prefers the item he/she gets in M over the item he/she gets in M'. A matching M is *more popular than* M' if the applicants that prefer M over M' outweigh those that prefer M' over M. Finally, a matching M is *weighted popular* if there is no matching M' more popular than M.

In the *weighted popular matching problem* it is necessary to determine if a given instance admits a popular matching, and if so, to produce one. In the *maximum weighted popular matching problem* it is necessary to find a popular matching of maximum cardinality, provided one exists.

Abraham et al. [2] gave the first polynomial time algorithms for the special case of these problems where the weights are uniform. Later, Mestre [8] introduced the weighted variant and developed polynomial time algorithms for it.

Key Results

Theorem 1 *The weighted popular matching and maximum weighted popular matching problems on instances with strictly ordered preferences can be solved in* $O(|X| + |E|)$ *time.*

Theorem 2 *The weighted popular matching and maximum weighted popular matching problems on instances with arbitrary preferences can be solved in* $O(\min\{k\sqrt{|X|}, |X|\}|E|)$ *time.*

Both results rely on an alternative easy-to-compute characterization of weighted popular matchings called *well-formed* matchings. It can be shown that every popular matching is well-formed. While in unweighted instances every well-formed matching is popular [2], in weighted instances there may be well-formed matchings that are not popular. These non-popular well-formed matchings can be weeded out by pruning certain bad edges that cannot be part of any popular matching. In other words, the instance can be pruned so that a matching is popular if and only if it is well-formed and is contained in the pruned instance [8].

Applications

Many real-life problems can be modeled using one-sided preferences. For example, the assignment of graduates to training positions [5], families to government-subsidized housing [10], students to projects [9], and Internet rental markets [1] such as Netflix where subscribers are assigned DVDs.

Furthermore, the weighted framework allows one to model the naturally occurring situation in which some subset of users has priority over the rest. For example, an Internet rental site may offer a "premium" subscription plan and promise priority over "regular" subscribers.

Cross References

- ▶ Ranked Matching
- ▶ Stable Marriage

Recommended Reading

1. Abraham, D.J., Chen, N., Kumar, V., Mirrokni, V.: Assignment problems in rental markets. In: Proceedings of the 2nd Workshop on Internet and Network Economics, Patras, December 15–17 2006
2. Abraham, D.J., Irving, R.W., Kavitha, T., Mehlhorn, K.: Popular matchings. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 424–432 (2005)
3. Abraham, D.J., Kavitha, T.: Dynamic matching markets and voting paths. In: Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT), pp. 65–76, Riga, July 6–8 2006
4. Gardenfors, P.: Match making: assignments based on bilateral preferences. Behav. Sci. **20**, 166–173 (1975)
5. Hylland, A., Zeeckhauser, R.: The efficent allocation of individuals to positions. J. Polit. Econ. **87**(2), 293–314 (1979)
6. Mahdian, M.: Random popular matchings. In: Proceedings of the 7th ACM Conference on Electronic Commerce (EC), pp. 238–242 Venice, July 10–14 2006
7. Manlove, D., Sng, C.: Popular matchings in the capacitated house allocation problem. In: Proceedings of the 14th Annual European Symposium on Algorithms (ESA), pp. 492–503 (2006)
8. Mestre, J.: Weighted popular matchings. In: Proceedings of the 16th International Colloquium on Automata, Languages, and Programming (ICALP), pp. 715–726 (2006)
9. Proll, L.G.: A simple method of assigning projects to students. Oper. Res. Q. **23**(23), 195–201 (1972)
10. Yuan, Y.: Residence exchange wanted: a stable residence exchange problem. Eur. J. Oper. Res. **90**, 536–546 (1996)

Weighted Random Sampling

2005; Efraimidis, Spirakis

Pavlos Efraimidis[1], Paul Spirakis[2]

[1] Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece

[2] Department of Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece

Keywords and Synonyms

Random number generation; Sampling

Problem Definition

The problem of random sampling without replacement (RS) calls for the selection of m distinct random items out of a population of size n. If all items have the same probability to be selected, the problem is known as uniform RS. Uniform random sampling in one pass is discussed in [1,6,11]. Reservoir-type uniform sampling algorithms over data streams are discussed in [12]. A parallel uniform random sampling algorithm is given in [10]. In weighted random sampling (WRS) the items are weighted and the probability of each item to be selected is determined by its relative weight. WRS can be defined with the following algorithm D:

Algorithm D, a definition of WRS

Input: A population V of n weighted items
Output: A set S with a WRS of size m
1: For $k = 1$ to m do
2: Let $p_i(k) = w_i / \sum_{s_j \in V-S} w_j$ be the probability of item v_i to be selected in round k
3: Randomly select an item $v_i \in V - S$ and insert it into S
4: End-For

Problem 1 (WRS)
INPUT: *A population V of n weighted items.*
OUTPUT: *A set S with a weighted random sample.*

The most important algorithms for WRS are the Alias Method, Partial Sum Trees and the Acceptance/Rejection method (see [9] for a summary of WRS algorithms). *None of these algorithms is appropriate for one-pass WRS.* In this work, an algorithm for WRS is presented. The algorithm is simple, very flexible, and solves the WRS problem over data streams. Furthermore, the algorithm admits parallel or distributed implementation. To the best knowledge of the entry authors, this is the first algorithm for WRS over data streams and for WRS in parallel or distributed settings.

Definitions

One-pass WRS is the problem of generating a weighted random sample in one-pass over a population. If additionally the population size is initially unknown (e. g. a data streams), the random sample can be generated with *reservoir sampling* algorithms. These algorithms keep an auxiliary storage, the reservoir, with all items that are candidates for the final sample.

Notation and Assumptions

The item weights are initially unknown, strictly positive reals. The population size is n, the size of the random sample is m and the weight of item v_i is w_i. The function $random(L, H)$ generates a uniform random number in (L, H). X denotes a random variable. Infinite precision arithmetic is assumed. Unless otherwise specified, all sampling problems are without replacement. Depending on the context, WRS is used to denote a weighted random sample or the operation of weighted random sampling.

Key Results

All the results with their proofs can be found in [4].

The crux of the WRS approach of this work is given with the following **algorithm A:**

Algorithm A

Input: A population V of n weighted items
Output: A WRS of size m
1: For each $v_i \in V$, $u_i = random(0, 1)$ and $k_i = u_i^{(1/w_i)}$
2: Select the m items with the largest keys k_i as a WRS

Theorem 1 *Algorithm A generates a WRS.*

A reservoir-type adaptation of algorithm A is the following **algorithm A-Res**:

Algorithm A with a Reservoir (A-Res)

Input: A population V of n weighted items
Output: A reservoir R with a WRS of size m
1: The first m items of V are inserted into R
2: For each item $v_i \in R$: Calculate a key $k_i = u_i^{(1/w_i)}$, where $u_i = random(0, 1)$
3: Repeat Steps 4–7 for $i = m + 1, m + 2, \ldots, n$
4: The smallest key in R is the current threshold T
5: For item v_i: Calculate a key $k_i = u_i^{(1/w_i)}$, where $u_i = random(0, 1)$
6: If the key k_i is larger than T, then:
7: The item with the minimum key in R is replaced by item v_i

Algorithm A-Res performs the calculations required by algorithm A and hence by Theorem 1 A-Res generates a WRS. The number of reservoir operations for algorithm A-Res is given by the following Proposition:

Theorem 2 *If A-Res is applied on n weighted items, where the weights $w_i > 0$ are independent random variables with a common continuous distribution, then the expected number of reservoir insertions (without the initial m insertions)*

is:

$$\sum_{i=m+1}^{n} P\,[\,\textit{item } i \textit{ is inserted into } S\,] = \sum_{i=m+1}^{n} \frac{m}{i} = O\left(m \cdot \log\left(\frac{n}{m}\right)\right).$$

Let S_w be the sum of the weights of the items that will be skipped by A-Res until a new item enters the reservoir. If T_w is the current threshold to enter the reservoir, then S_w is a continuous random variable that follows an exponential distribution. Instead of generating a key for every item, it is possible to generate random jumps that correspond to the sum S_w. Similar techniques have been applied for uniform random sampling (see for example [3]). The following algorithm A-ExpJ is an exponential jumps-type adaptation of algorithm A:

Algorithm A with exponential jumps (A-ExpJ)

Input: A population V of n weighted items
Output: A reservoir R with a WRS of size m
1: The first m items of V are inserted into R
2: For each item $v_i \in R$: Calculate a key $k_i = u_i^{(1/w_i)}$, where $u_i = random(0, 1)$
3: The threshold T_w is the minimum key of R
4: Repeat Steps 5–10 until the population is exhausted
5: Let $r = random(0, 1)$ and $X_w = \log(r)/\log(T_w)$
6: From the current item v_c skip items until item v_i, such that:
7: $w_c + w_{c+1} + \cdots + w_{i-1} < X_w \leq w_c + w_{c+1} + \cdots + w_{i-1} + w_i$
8: The item in R with the minimum key is replaced by item v_i
9: Let $t_w = {T_w}^{w_i}$, $r_2 = random(t_w, 1)$ and v_i's key: $k_i = {r_2}^{(1/w_i)}$
10: The new threshold T_w is the new minimum key of R

Theorem 3 *Algorithm A-ExpJ generates a WRS.*

The number of exponential jumps of A-ExpJ is given by Proposition 2. Hence algorithm A-ExpJ reduces the number of random variates that have to be generated from $O(n)$ (for A-Res) to $O(m \log(n/m))$. Since generating high-quality random variates can be a costly operation this is a significant improvement for the complexity of the sampling algorithm.

Applications

Random sampling is a fundamental problem in computer science with applications in many fields including databases (see [5,9] and the references therein), data mining, and approximation algorithms and randomized algorithms [7]. Consequently, algorithm A for WRS is a general tool that can find applications in the design of randomized algorithms. For example, algorithm A can be used within approximation algorithms for the k-Median [7].

The reservoir based versions of algorithm A, A-Res and A-ExpJ, have very small requirements for auxiliary storage space (m keys organized as a heap) and during the sampling process their reservoir continuously contains a weighted random sample that is valid for the already processed data. This makes the algorithms applicable to the emerging area of algorithms for processing data streams([2,8]).

Algorithms A-Res and A-ExpJ can be used for weighted random sampling with replacement from data streams. In particular, it is possible to generate a weighted random sample with replacement of size k with A-Res or A-ExpJ, by running concurrently, in one pass, k instances of A-Res or A-ExpJ respectively. Each algorithm instance must be executed with a trivial reservoir of size 1. At the end, the union of all reservoirs is a WRS with replacement.

URL to Code

The algorithms presented in this work are easy to implement. An experimental implementation in Java can be found at: http://utopia.duth.gr/~pefraimi/projects/WRS/index.html

Cross References

► Online Paging and Caching
► Randomization in Distributed Computing

Recommended Reading

1. Ahrens, J.H., Dieter, U.: Sequential random sampling. ACM Trans. Math. Softw. **11**, 157–169 (1985)
2. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 1–16. ACM Press (2002)
3. Devroye, L.: Non-uniform Random Variate Generation. Springer, New York (1986)
4. Efraimidis, P., Spirakis, P.: Weighted Random Sampling with a reservoir. Inf. Process. Lett. J. **97**(5), 181–185 (2006)
5. Jermaine, C., Pol, A., Arumugam, S.: Online maintenance of very large random samples. In: SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, New York, pp. 299–310. ACM Press (2004)
6. Knuth, D.: The Art of Computer Programming, vol. 2 : Seminumerical Algorithms, 2nd edn. Addison-Wesley Publishing Company, Reading (1981)

7. Lin, J.-H., Vitter, J.: ϵ-approximations with minimum packing constraint violation. In: 24th ACM STOC, pp. 771–782 (1992)
8. Muthukrishnan, S.: Data streams: Algorithms and applications. Found. Trends Theor. Comput. Sci. **1**, pp.1–126 (2005)
9. Olken, F.: Random Sampling from Databases. Ph. D. thesis, Department of Computer Science, University of California, Berkeley (1993)
10. Rajan, V., Ghosh, R., Gupta, P.: An efficient parallel algorithm for random sampling. Inf. Process. Lett. **30**, 265–268 (1989)
11. Vitter, J.: Faster methods for random sampling. Commun. ACM **27**, 703–718 (1984)
12. Vitter, J.: Random sampling with a reservoir. ACM Trans. Math. Softw. **11**, 37–57 (1985)

Well Separated Pair Decomposition

2003; Gao, Zhang

Jie Gao[1], Li Zhang[2]
[1] Department of Computer Science, Stony Brook University, Stony Brook, NY, USA
[2] HP Labs, Palo Alto, CA, USA

Keywords and Synonyms

Proximity algorithms for growth-restricted metrics

Problem Definition

Well-separated pair decomposition, introduced by Callahan and Kosaraju [3], has found numerous applications in solving proximity problems for points in the Euclidean space. A pair of point sets (A, B) is *c-well-separated* if the distance between A and B is at least c times the diameters of both A and B. A well-separated pair decomposition of a point set consists of a set of well-separated pairs that "cover" all the pairs of distinct points, i. e., any two distinct points belong to the different sets of some pair. Callahan and Kosaraju [3] showed that for any point set in a Euclidean space and for any constant $c \geq 1$, there always exists a c-well-separated pair decomposition (c-WSPD) with linearly many pairs. This fact has been very useful for obtaining nearly linear time algorithms for many problems, such as computing k-nearest neighbors, N-body potential fields, geometric spanners, approximate minimum spanning trees, etc. Well-separated pair decomposition has also been shown to be very useful for obtaining efficient dynamic, parallel, and external memory algorithms.

The definition of well-separated pair decomposition can be naturally extended to any metric space. However, a general metric space may not admit a well-separated pair decomposition with a subquadratic size. Indeed, even for the metric induced by a star tree with unit weight on each edge,[1] any well-separated pair decomposition requires quadratically many pairs. This makes the well-separated pair decomposition useless for such a metric. However, it has been shown that for the unit disk graph metric, there do exist well-separated pair decompositions with almost linear size, and therefore many proximity problems under the unit disk graph metric can be solved efficiently.

Unit-Disk Graphs [4]

Denote by $\mathrm{d}(\cdot,\cdot)$ the Euclidean metric. For a set of points S in the plane, the unit-disk graph $I(S) = (S, E)$ is defined to be the weighted graph where an edge $e = (p, q)$ is in the graph if $\mathrm{d}(p, q) \leq 1$, and the weight of e is $\mathrm{d}(p, q)$. Likewise, one can define the unit-ball graph for points in higher dimensions.

Unit-disk graphs have been used extensively to model the communication or influence between objects [9,12] and have been studied in many different contexts [4,10]. For an example, wireless ad hoc networks can be modeled by unit-disk graphs [6], as two wireless nodes can directly communicate with each other only if they are within a certain distance. In unsupervised learning, for a dense sampling of points from some unknown manifold, the length of the shortest path on the unit-ball graph is a good approximation of the geodesic distance on the underlying (unknown) manifold if the radius is chosen appropriately [14,5]. By using well-separated pair decomposition, one can encode the all-pairs distances approximately by a compact data structure that supports approximate distance queries in $O(1)$ time.

Metric Space

Suppose that (S, π) is a metric space where S is a set of elements and π the distance function defined on $S \times S$. For any subset $S_1 \subseteq S$, the *diameter* $D_\pi(S_1)$ (or $D(S_1)$ when π is clear from the context) of S is defined to be $\max_{s_1,s_2 \in S_1} \pi(s_1, s_2)$. The *distance* $\pi(S_1, S_2)$ between two sets $S_1, S_2 \subseteq S$ is defined to be $\min_{s_1 \in S_1, s_2 \in S_2} \pi(s_1, s_2)$.

Well-Separated Pair Decomposition

For a metric space (S, π), two nonempty subsets $S_1, S_2 \subseteq S$ are called *c-well-separated* if $\pi(S_1, S_2) \geq c \cdot \max(D_\pi(S_1), D_\pi(S_2))$.

Following the definition in [3], for any two sets A and B, a set of pairs $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$, where $P_i = (A_i, B_i)$, is called a *pair decomposition* of (A, B) (or of A if $A = B$) if

[1] A metric induced by a graph (with positive edge weights) is the shortest-path distance metric of the graph.

- For all the i's, $A_i \subseteq A$, and $B_i \subseteq B$.
- $A_i \cap B_i = \emptyset$.
- For any two elements $a \in A$ and $b \in B$, there exists a unique i such that $a \in A_i$, and $b \in B_i$. Call (a, b) is *covered* by the pair (A_i, B_i).

If in addition, every pair in $\mathcal{P}$ is c-well-separated, $\mathcal{P}$ is called a *c-well-separated pair decomposition* (or c-WSPD for short). Clearly, any metric space admits a c-WSPD with quadratic size by using the trivial family that contains all the pairwise elements.

Key Results

In [7], it was shown that for the metric induced by the unit-disk graph on n points and for any constant $c \geq 1$, there does exist a c-WSPD with $O(n \log n)$ pairs, and such a decomposition can be computed in $O(n \log n)$ time. It was also shown that the bounds can be extended to higher dimensions. The following theorems state the key results for two and higher dimensions.

Theorem 1 *For any set S of n points in the plane and any $c \geq 1$, there exists a c-WSPD $\mathcal{P}$ of S under the unit disk graph metric where $\mathcal{P}$ contains $O(c^4 n \log n)$ pairs and can be computed in $O(c^4 n \log n)$ time.*

Theorem 2 *For any set S of n points in $\mathbb{R}^k$, for $k \geq 3$, and for any constant $c \geq 1$, there exists a c-WSPD $\mathcal{P}$ of S under the unit ball graph metric where $\mathcal{P}$ contains $O(n^{2-2/k})$ pairs and can be constructed in $O(n^{4/3}$ polylog $n)$ time for $k = 3$ and in $O(n^{2-2/k})$ time for $k \geq 4$.*

The difficulty in obtaining a well-separated pair decomposition for the unit disk graph metric is that two points that are close in space are not necessarily close under the graph metric. The above bounds are first shown for the point set with constant-bounded density, i. e., a point set where any unit disk covers only a constant number of points in the set. The upper bound on the number of pairs is obtained by using a packing argument similar to the one used in [8].

For a point set with unbounded density, one applies a clustering technique similar to the one used in [6] to the point set and obtains a set of "clusterheads" with a bounded density. Then the result for bounded density is applied to those clusterheads. Finally, the well-separated pair decomposition is obtained by combining the well-separated pair decomposition for the bounded density point sets and for the Euclidean metric. The number of pairs is dominated by the number of pairs constructed for a constant density set, which is in turn dominated by the bound given by the packing argument. It has been shown that the bounds on the number of pairs is tight for $k \geq 3$.

Applications

For a pair of well-separated sets, the distance between two points from different sets can be approximated by the "distance" between the two sets or the distance between any pair of points in different sets. In other words, a well-separated pair decomposition can be thought of as a compressed representation to approximate the $\Theta(n^2)$ pairwise distances. Many problems that require the pairwise distances to be checked can therefore be approximately solved by examining those distances between the well-separated pairs of sets. When the size of the well-separated pair decomposition is subquadratic, it often results in more efficient algorithms than examining all the pairwise distances. Indeed, this is the intuition behind many applications of the geometric well-separated pair decomposition. By using the same intuition, one can apply the well-separated pair decomposition in several proximity problems under the unit disk graph metric.

Suppose that (S, d) is a metric space. Let $S_1 \subseteq S$. Consider the following natural proximity problems.

- **Furthest neighbor, diameter, center.** The furthest neighbor of $p \in S_1$ is the point in S_1 that maximizes the distance to p. Related problems include computing the *diameter*, the maximum pairwise shortest distance for points in S_1, and the *center*, the point that minimizes the maximum distance to all the other points.
- **Nearest neighbor, closest pair.** The nearest neighbor of $p \in S_1$ is the point in S_1 with the minimum distance to p. Related problems include computing the *closest pair*, the pair with the minimum shortest distance, and the *bichromatic closest pair*, the pair that minimizes the distance between points from two different sets.
- **Median.** The median of S is the point in S that minimizes the average (or total) distance to all the other points.
- **Stretch factor.** For a graph G defined on S, its stretch factor with respect to the unit disk graph metric is defined to be the maximum ratio $\pi_G(p, q)/\pi(p, q)$, where π_G, π are the distances induced by G and by the unit-disk graph, respectively.

All the above problems can be solved or approximated efficiently for points in the Euclidean space. However, for the metric induced by a graph, even for planar graphs, very little is known besides solving the expensive all-pairs shortest-path problem. For computing the diameter, there is a simple linear-time method that achieves a 2-approx-

imation[2] and a 4/3-approximate algorithm with running time $O(m\sqrt{n \log n} + n^2 \log n)$, for a graph with n vertices and m edges, by Aingworth et al. [1].

By using the well-separated pair decomposition, Gao and Zhang [7] showed that one can obtain better approximation algorithms for the above proximity problems for the unit disk graph metric. Specifically, one can obtain almost linear-time algorithms for computing the 2.42-approximation and $O(n\sqrt{n \log n}/\varepsilon^3)$ time algorithms for computing the $(1 + \varepsilon)$-approximation for any $\varepsilon > 0$. In addition, the well-separated pair decomposition can be used to obtain an $O(n \log n/\varepsilon^4)$ space distance oracle so that any $(1 + \varepsilon)$ distance query in the unit-disk graph can be answered in $O(1)$ time.

The bottleneck of the above algorithms turns out to be computing the approximation of the shortest path distances between $O(n \log n)$ pairs. The algorithm in [7] only constructs well-separated pair decompositions without computing a good approximation of the distances. The approximation ratio and the running time are dominated by that of the approximation algorithms used to estimate the distance between each pair in the well-separated pair decomposition. Once the distance estimation has been made, the rest of the computation only takes almost linear time.

For a general graph, it is unknown whether $O(n \log n)$ pairs shortest-path distances can be computed significantly faster than all-pairs shortest-path distances. For a planar graph, one can compute the $O(n \log n)$ pairs shortest-path distances in $O(n\sqrt{n \log n})$ time by using separators with $O(\sqrt{n})$ size [2]. This method extends to the unit-disk graph with constant bounded density since such graphs enjoy a separator property similar to that of planar graphs [13]. As for approximation, Thorup [15] recently discovered an algorithm for planar graphs that can answer any $(1 + \varepsilon)$-shortest-distance query in $O(1/\varepsilon)$ time after almost linear time preprocessing. Unfortunately, Thorup's algorithm uses balanced shortest-path separators in planar graphs which do not obviously extend to the unit-disk graphs. On the other hand, it is known that there does exist a planar 2.42-spanner for a unit-disk graph [11]. By applying Thorup's algorithm to that planar spanner, one can compute the 2.42-approximate shortest-path distance for $O(n \log n)$ pairs in almost linear time.

[2]Select an arbitrary node v and compute the shortest-path tree rooted at v. Suppose that the furthest node from v is distance D away. Then the diameter of the graph is no longer than $2D$, by triangle inequality.

Open Problems

The most notable open problem is the gap between $\Omega(n)$ and $O(n \log n)$ on the number of pairs needed in the plane. Also, the time bound for $(1 + \varepsilon)$-approximation is still about $\widetilde{O}(n\sqrt{n})$ due to the lack of efficient methods for computing the $(1 + \varepsilon)$-approximate shortest path distances between $O(n)$ pairs of points. Any improvement to the algorithm for that problem will immediately lead to improvement to all the $(1 + \varepsilon)$-approximate algorithms presented in this chapter.

Cross References

- ▶ Applications of Geometric Spanner Networks
- ▶ Separators in Graphs
- ▶ Sparse Graph Spanners
- ▶ Well Separated Pair Decomposition for Unit–Disk Graph

Recommended Reading

1. Aingworth, D., Chekuri, C., Motwani, R.: Fast estimation of diameter and shortest paths (without matrix multiplication). In: Proc. 7th ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 547–553
2. Arikati, S.R., Chen, D.Z., Chew, L.P., Das, G., Smid, M.H.M., Zaroliagis, C.D: Planar spanners and approximate shortest path queries among obstacles in the plane. In: Díaz, J., Serna, M. (eds.) Proc. of 4th Annual European Symposium on Algorithms, 1996, pp. 514–528
3. Callahan, P.B., Kosaraju, S. R.: A decomposition of multidimensional point sets with applications to *k*-nearest-neighbors and *n*-body potential fields. J. ACM **42**, 67–90 (1995)
4. Clark, B.N., Colbourn, C.J., Johnson, D.S.: Unit disk graphs. Discret. Math. **86**, 165–177 (1990)
5. Fischl, B., Sereno, M., Dale, A.: Cortical surface-based analysis II: Inflation, flattening, and a surface-based coordinate system. NeuroImage **9**, 195–207 (1999)
6. Gao, J., Guibas, L.J., Hershberger, J., Zhang, L., Zhu, A.: Geometric spanners for routing in mobile networks. IEEE J. Sel. Areas Commun. Wirel. Ad Hoc Netw. (J-SAC), **23**(1), 174–185 (2005)
7. Gao, J., Zhang, L.: Well-separated pair decomposition for the unit-disk graph metric and its applications. In: Proc. of 35th ACM Symposium on Theory of Computing (STOC'03), 2003, pp. 483–492
8. Guibas, L., Ngyuen, A., Russel, D., Zhang, L.: Collision detection for deforming necklaces. In: Proc. 18th ACM Symposium on Computational Geometry, 2002, pp. 33–42
9. Hale, W. K.: Frequency assignment: Theory and applications. Proc. IEEE. **68**(12), 1497–1513 (1980)
10. H.B.H. III, Marathe, M.V., Radhakrishnan, V., Ravi, S.S., Rosenkrantz, D.J., Stearns, R.E.: NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. J. Algorithms **26**(2), 238–274 (1998)
11. Li, X.Y., Calinescu, G., Wan, P.J.: Distributed Construction of a Planar Spanner and Routing for Ad Hoc Wireless Networks. In: IEEE INFOCOM 2002, New York, NY, 23–27 June 2002

12. Mead, C.A., Conway, L.: Introduction to VLSI Systems. Addison-Wesley, (1980)
13. Miller, G.L., Teng, S.H., Vavasis, S.A.: An unified geometric approach to graph separators. In: Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci. 1991, pp. 538–547
14. Tenenbaum, J., de Silva, V., Langford, J.: A global geometric framework for nonlinear dimensionality reduction. Science **290**, 22 (2000)
15. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. In: Proc. 42nd IEEE Symposium on Foundations of Computer Science, 2001, pp. 242–251

Well Separated Pair Decomposition for Unit–Disk Graph

1995; Callahan, Kosaraju

Rolf Klein
Institute of Computer Science, University of Bonn, Bonn, Germany

Keywords and Synonyms

Clustering

Problem Definition

Notations

Given a finite point set A in $\mathbb{R}^d$, its *bounding box* $R(A)$ is the d-dimensional hyper-rectangle $[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$ that contains A and has minimum extension in each dimension.

Two point sets A, B are said to be *well-separated* with respect to a separation parameter $s > 0$ if there exist a real number $r > 0$ and two d-dimensional spheres C_A and C_B of radius r each, such that the following properties are fulfilled.

1. $C_A \cap C_B = \emptyset$
2. C_A contains the bounding box $R(A)$ of A
3. C_B contains the bounding box $R(B)$ of B
4. $|C_A C_B| \geq s \cdot r$.

Here $|C_A C_B|$ denotes the smallest Euclidean distance between two points of C_A and C_B, respectively. An example is depicted in Fig. 1. Given the bounding boxes $R(A)$, $R(B)$, it takes time only $O(d)$ to test if A and B are well-separated with respect to s.

Two points of the same set, A or B, have a Euclidean distance at most $2/s$ times the distance any pair $(a, b) \in A \times B$ can have. Also, any two such pairs $(a, b), (a', b')$ differ in their distances $|a - b|, |a' - b'|$ by a factor of at most $1 + 4/s$.

Given a set S of n points in $\mathbb{R}^d$, a *well-separated pair decomposition* of S with respect to separation parameter s is a sequence $(A_1, B_1), (A_2, B_2), \ldots, (A_m, B_m)$ where

1. $A_i, B_i \subset S$, for $i = 1 \ldots m$
2. A_i and B_i are well-separated with respect to s, for $i = 1 \ldots m$
3. for all points $a, b \in S$, $a \neq b$, there exists a unique index i in $1 \ldots m$ such that $a \in A_i$ and $b \in B_i$, or $b \in A_i$ and $a \in B_i$ hold

Obviously, each set $S = \{s_1, \ldots, s_n\}$ possesses a well-separated pair decomposition. One can simply use all singleton pairs $(\{s_i\}, \{s_j\})$ where $i < j$. The question is if decompositions consisting of fewer than $O(n^2)$ many pairs exist, and how to construct them efficiently.

Key Results

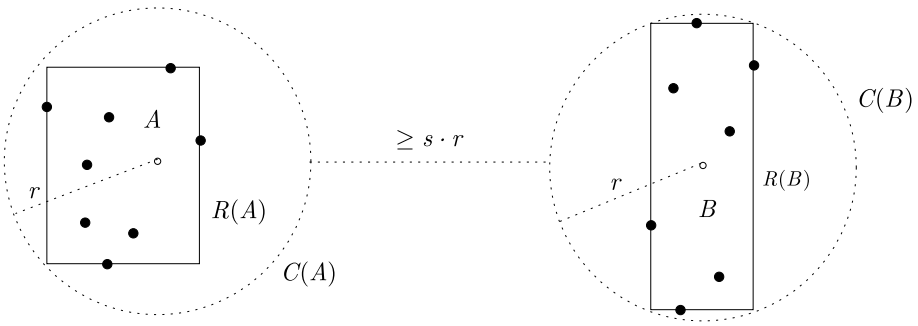
In fact, the following result has been shown by Callahan and Kosaraju [1,2].

Theorem 1 *Given a set S of n points in $\mathbb{R}^d$ and a separation parameter s, there exists a well-separated pair decomposition of S with respect to s, that consists of $O(s^d d^{d/2} n)$ many pairs (A_i, B_i). It can be constructed in time $O(dn \log n + s^d d^{d/2+1} n)$.*

Thus, if dimension d and separation parameter s are fixed – which is the case in many applications – then the number of pairs is in $O(n)$, and the decomposition can be computed in time $O(n \log n)$.

The main tool in constructing the well-separated pair decomposition is the *split tree* $T(S)$ of S. The root, r, of $T(S)$ contains the bounding box $R(S)$ of S. Its two child nodes are obtained by cutting through the middle of the longest dimension of $R(S)$, using an orthogonal hyperplane. It splits S into two subsets S_a, S_b, whose bounding boxes $R(S_a)$ and $R(S_b)$ are stored at the two children a and b of root r. This process continues until only one point of S remains in each subset. These singleton sets form the leaves of $T(S)$. Clearly, the split tree $T(S)$ contains $O(n)$ many nodes. It need not be balanced, but it can be constructed in time $O(dn \log n)$.

A well-separated pair decomposition of S, with respect to a given separation parameter s, can now be obtained from $T(S)$ in the following way. For each internal node of $T(S)$ with children v and w the following recursive procedure FindPairs(v,w) is called. If S_v and S_w are well-separated then the pair (S_v, S_w) is reported. Otherwise, one may assume that the longest dimension of $R(S_v)$ exceeds in length the longest dimension of $R(S_w)$, and that v_l, v_r are the child nodes of v in $T(S)$. Then, FindPairs(v_l, w) and FindPairs(v_r, w) are invoked.



Well Separated Pair Decomposition for Unit–Disk Graph, Figure 1
The sets A, B are well-separated with respect to s

The total number of procedure calls is bounded by the number of well-separated pairs reported, which can be shown to be in $O(s^d d^{d/2} n)$ by a packing argument. However, the total size of all sets A_i, B_i in the decomposition is in general quadratic in n.

Applications

From now on the dimension d is assumed to be a constant. The well-separated pair decomposition can be used in efficiently solving proximity problems for points in $\mathbb{R}^d$.

Theorem 2 *Let S be a set of n points in $\mathbb{R}^d$. Then a closest pair in S can be found in optimal time $O(n \log n)$.*

Indeed, let $q \in S$ be a nearest neighbor of $p \in S$. One can construct a well-separated pair decomposition with separation parameter $s > 2$ in time $O(n \log n)$, and let (A_i, B_i) be the pair where $p \in A_i$ and $q \in B_i$. If there were another point p' of S in A_i, one would obtain $|pp'| \le 2/s \cdot |pq| < |pq|$, which is impossible. Hence, A_i is a singleton set. If (p, q) is a closest pair in S then B_i must be singleton, too. Therefore, a closest pair can be found by inspecting all singleton pairs among the $O(n)$ many pairs of the well-separated pair decomposition.

With more effort, the following generalization can been shown.

Theorem 3 *Let S be a set of n points in $\mathbb{R}^d$, and let $k \le n$. Then for each $p \in S$ its k nearest neighbors in S can be computed in total time $O(n \log n + nk)$. In particular, for each point in S can a nearest neighbor in S be computed in optimal time $O(n \log n)$.*

In dimension $d = 2$ one would typically use the *Voronoi diagram* for solving these problems. But as the complexity of the Voronoi diagram of n points can be as large as $n^{\lfloor d/2 \rfloor}$, the well-separated pair decomposition is much more convenient to use in higher dimensions.

A major application of the well-separated pair decomposition is the construction of good *spanners* for a given point set S. A spanner of S of dilation t is a geometric network N with vertex set S such that, for any two vertices $p, q \in S$, the Euclidean length of a shortest path connecting p and q in N is at most t times the Euclidean distance $|pq|$.

Theorem 4 *Let S be a set of n points in $\mathbb{R}^d$, and let $t>1$. Then a spanner of S of dilation t containing $O(s^d n)$ edges can be constructed in time $O(s^d n + n \log n)$, where $s = 4(t+1)(t-1)$.*

Indeed, if one edge (a_i, b_i) is chosen from each pair (A_i, B_i) of a well-separated pair decomposition of S with respect to s, these edges form a t-spanner of S, as can be shown by induction on the rank of each pair $(p, q) \in S^2$ in the list of all such pairs, sorted by distance.

Since spanners have many interesting applications of their own, several articles of this encyclopedia are devoted to this topic.

Open Problems

An important open question is which metric spaces admit well-separated pair decompositions. It is easy to see that the packing arguments used in the Euclidean case carry over to the case of convex distance functions in $\mathbb{R}^d$. More generally, Talwar [6] has shown how to compute well-separated pair decompositions for point sets of bounded aspect ratio in metric spaces of bounded doubling dimension.

On the other hand, for the metric induced by a disk graph in $\mathbb{R}^2$, a quadratic number of pairs may be necessary in the well-separated pair decomposition. (In a disk graph, each point $p \in S$ is center of a disk D_p of radius r_p. Two points p, q are connected by an edge if and only if $D_p \cap D_q \neq \emptyset$. The metric is defined by Euclidean short-

est path length in the resulting graph. If this graph is a star with rays of identical length, a well-separated pair decomposition with respect to $s > 4$ must consist of singleton pairs.) Even for a unit disk graph, $\Omega(n^{2-2/d})$ many pairs may be necessary for points in $\mathbb{R}^d$, as Gao and Zhang [4] have shown.

Cross References

- ▶ Applications of Geometric Spanner Networks
- ▶ Geometric Spanners
- ▶ Planar Geometric Spanners

Recommended Reading

1. Callahan, P.: Dealing with Higher Dimensions: The Well-Separated Pair Decomposition and Its Applications. Ph. D. Thesis, The Johns Hopkins University, USA (1995)
2. Callahan, P.B., Kosaraju, S.R.: A Decomposition of Multidimensional Point Sets with Applications to k-Nearest Neighbors and n-Body Potential Fields. J. ACM **42**(1), 67–90 (1995)
3. Eppstein, D.: Spanning Trees and Spanners. In: Sack, J.R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 425–461. Elsevier, Amsterdam (1999)
4. Ghao, J., Zhang, L.: Well-Separated Pair Decomposition for the Unit Disk Graph Metric and its Applications. SIAM J. Comput. **35**(1), 151–169 (2005)
5. Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press, New York (2007)
6. Talwar, K.: Bypassing the Embedding: Approximation Schemes and Compact Representations for Low Dimensional Metrics. In: Proceedings of the thirty-sixth Annual ACM Symposium on Theory of Computing (STOC'04), pp. 281–290 (2004)

Whole Genome Assemble

- ▶ Multiplex PCR for Gap Closing (Whole-genome Assembly)

Wireless Networks

- ▶ Broadcasting in Geometric Radio Networks
- ▶ Deterministic Broadcasting in Radio Networks
- ▶ Randomized Gossiping in Radio Networks

Wire Sizing

1999; Chu, Wong

Chris Chu
Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, USA

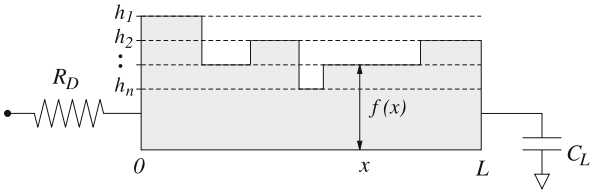
Keywords and Synonyms

Wire tapering

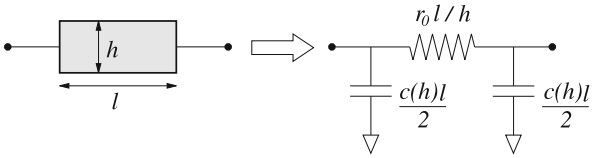
Problem Definition

The problem is about minimizing the delay of an interconnect wire in a Very Large Scale Integration (VLSI) circuit by changing (i. e., sizing) the width of the wire. The delay of interconnect wire has become a dominant factor in determining VLSI circuit performance for advanced VLSI technology. Wire sizing has been shown to be an effective technique to minimize the interconnect delay. The work of Chu and Wong [5] shows that the wire sizing problem can be transformed into a convex quadratic program. This quadratic programming approach is very efficient and can be naturally extended to simultaneously consider buffer insertion, which is another popular interconnect delay minimization technique. Previous approaches apply either a dynamic programming approach [13], which is computationally more expensive, or an iterative greedy approach [2,7], which is hard to combine with buffer insertion.

The wire sizing problem is formulated as follows and is illustrated in Fig. 1. Consider a wire of length L. The wire is connecting a driver with driver resistance R_D to a load with load capacitance C_L. In addition, there is a set $H = \{h_1, \ldots, h_n\}$ of n wire widths allowed by the fabrication technology. Assume $h_1 > \cdots > h_n$. The wire sizing problem is to determine the wire width function $f(x) : [0, L] \to H$ so that the delay for a signal to travel from the driver through the wire to the load is minimized.



Wire Sizing, Figure 1
Illustration of the wire sizing problem



Wire Sizing, Figure 2
The model of a wire segment of length *l* and width *h* by a π-type RC circuit

As in most previous works on wire sizing, the work of Chu and Wong uses the Elmore delay model to compute the delay. The Elmore delay model is a delay model for RC circuits (i. e., circuits consisting of resistors and capacitors). The Elmore delay for a signal path is equal to the sum of the delays associated with all resistors along the path, where the delay associated with each resistor is equal to its resistance times its total downstream capacitance. For a wire segment of length l and width h, its resistance is $r_0 l/h$ and its capacitance is $c(h)l$, where r_0 is the wire sheet resistance and $c(h)$ is the unit length wire capacitance. $c(h)$ is an increasing function in practice. The wire segment can be modeled as a π-type RC circuit as shown in Fig. 2.

Key Results

Lemma 1 *The optimal wire width function $f(x)$ is a monotonically decreasing function.*

Lemma 1 above can be used to greatly simplify the wire sizing problem. It implies that an optimally-sized wire can be divided into n segments such that the width of ith segment is h_i. The length of each segment is to be determined. The simplified problem is illustrated in Fig. 3.

Lemma 2 *For the wire in Fig. 3, the Elmore delay is*

$$D = \frac{1}{2}\boldsymbol{l}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{l} + \boldsymbol{\rho}^{\mathrm{T}}\boldsymbol{l} + R_D C_L$$

where

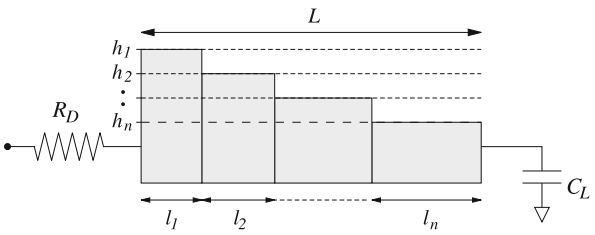
$$\boldsymbol{\Phi} = \begin{pmatrix} c(h_1)r_0/h_1 & c(h_2)r_0/h_1 & c(h_3)r_0/h_1 & \cdots & c(h_n)r_0/h_1 \\ c(h_2)r_0/h_1 & c(h_2)r_0/h_2 & c(h_3)r_0/h_2 & \cdots & c(h_n)r_0/h_2 \\ c(h_3)r_0/h_1 & c(h_3)r_0/h_2 & c(h_3)r_0/h_3 & \cdots & c(h_n)r_0/h_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c(h_n)r_0/h_1 & c(h_n)r_0/h_2 & c(h_n)r_0/h_3 & \cdots & c(h_n)r_0/h_n \end{pmatrix}$$

$$\boldsymbol{\rho} = \begin{pmatrix} R_D c(h_1) + C_L r_0/h_1 \\ R_D c(h_2) + C_L r_0/h_2 \\ R_D c(h_3) + C_L r_0/h_3 \\ \vdots \\ R_D c(h_n) + C_L r_0/h_n \end{pmatrix} \text{ and } \boldsymbol{l} = \begin{pmatrix} l_1 \\ l_2 \\ l_3 \\ \vdots \\ l_n \end{pmatrix}.$$

So the wire sizing problem can be written as the following quadratic program:

$$\mathcal{WS}: \quad \begin{array}{ll} \text{minimize} & \frac{1}{2}\boldsymbol{l}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{l} + \boldsymbol{\rho}^{\mathrm{T}}\boldsymbol{l} \\ \text{subject to} & l_1 + \cdots + l_n = L \\ & l_i \geq 0 \text{ for } 1 \leq i \leq n\,. \end{array}$$

Quadratic programming is NP-hard in general. In order to solve $\mathcal{WS}$ efficiently, some properties of the Hessian matrix $\boldsymbol{\Phi}$ are explored.



Wire Sizing, Figure 3
Illustration of the simplified wire sizing problem

Definition 1 (Symmetric Decomposable Matrix) Let $\boldsymbol{Q} = (q_{ij})$ be an $n \times n$ symmetric matrix. If for some $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)^{\mathrm{T}}$ and $\boldsymbol{\nu} = (\nu_1, \ldots, \nu_n)^{\mathrm{T}}$ such that $0 < \alpha_1 < \cdots < \alpha_n$, $q_{ij} = q_{ji} = \alpha_i \nu_i \nu_j$ for $i \leq$ j, then $\boldsymbol{Q}$ is called a symmetric decomposable matrix. Let $\boldsymbol{Q}$ be denoted as $SDM(\boldsymbol{\alpha}, \boldsymbol{\nu})$.

Lemma 3 *If $\boldsymbol{Q}$ is symmetric decomposable, then $\boldsymbol{Q}$ is positive definite.*

Lemma 4 *$\boldsymbol{\Phi}$ in $\mathcal{WS}$ is symmetric decomposable.*

Lemma 3 and Lemma 4 imply that the Hessian matrix $\boldsymbol{\Phi}$ of $\mathcal{WS}$ is positive definite. Hence, the problem $\mathcal{WS}$ is a *convex* quadratic program and is solvable in polynomial time [12].

The work of Chu and Wong proposes to solve $\mathcal{WS}$ by active set method. The active set method transforms a problem with some inequality constraints into a sequence of problems with only equality constraints. The method stops when the solution of the transformed problem satisfies both the feasibility and optimality conditions of the original problem. For the problem $\mathcal{WS}$, the active set method keeps track of an active set $\mathcal{A}$ in each iteration. The method sets $l_j = 0$ for all $j \in \mathcal{A}$ and ignores the constraints $l_j \geq 0$ for all $j \notin \mathcal{A}$. If $\{j_1, \ldots, j_r\} = \{1, \ldots, n\} - \mathcal{A}$, then $\mathcal{WS}$ is transformed into the following equality-constrained wire sizing problem:

$$\mathcal{ECWS}: \quad \begin{array}{ll} \text{minimize} & \frac{1}{2}\boldsymbol{l}_{\mathcal{A}}^{\mathrm{T}}\boldsymbol{\Phi}_{\mathcal{A}}\boldsymbol{l}_{\mathcal{A}} + \boldsymbol{\rho}_{\mathcal{A}}^{\mathrm{T}}\boldsymbol{l}_{\mathcal{A}} \\ \text{subject to} & \boldsymbol{\Gamma}_{\mathcal{A}}\boldsymbol{l}_{\mathcal{A}} = L \end{array}$$

where $\boldsymbol{l}_{\mathcal{A}} = (l_{j_1}, \ldots, l_{j_r})^{\mathrm{T}}$, $\boldsymbol{\Gamma}_{\mathcal{A}} = (1\ 1\ \cdots\ 1)$, $\boldsymbol{\rho}_{\mathcal{A}} = (R_D c(h_{j_1}) + C_L r_0/h_{j_1}, \ldots, R_D c(h_{j_r}) + C_L r_0/h_{j_r})^{\mathrm{T}}$, and $\boldsymbol{\Phi}_{\mathcal{A}}$ is the symmetric decomposable matrix corresponding to $\mathcal{A}$ (i. e., $\boldsymbol{\Phi}_{\mathcal{A}} = SDM(\boldsymbol{\alpha}_{\mathcal{A}}, \boldsymbol{\nu}_{\mathcal{A}})$ with $\boldsymbol{\alpha}_{\mathcal{A}} = (r_0/c(h_{j_1})h_{j_1}, \ldots, r_0/c(h_{j_r})h_{j_r})^{\mathrm{T}}$ and $\boldsymbol{\nu}_{\mathcal{A}} = (c(h_{j_1}), \ldots, c(h_{j_r}))^{\mathrm{T}}$).

Lemma 5 *The solution of $\mathcal{ECWS}$ is:*

$$\begin{aligned} \lambda_{\mathcal{A}} &= -(\boldsymbol{\Gamma}_{\mathcal{A}}\boldsymbol{\Phi}_{\mathcal{A}}^{-1}\boldsymbol{\Gamma}_{\mathcal{A}}^{\mathrm{T}})^{-1}(\boldsymbol{\Gamma}_{\mathcal{A}}\boldsymbol{\Phi}_{\mathcal{A}}^{-1}\boldsymbol{\rho}_{\mathcal{A}} + L) \\ \boldsymbol{l}_{\mathcal{A}} &= -\boldsymbol{\Phi}_{\mathcal{A}}^{-1}\boldsymbol{\Gamma}_{\mathcal{A}}^{\mathrm{T}}\lambda_{\mathcal{A}} - \boldsymbol{\Phi}_{\mathcal{A}}^{-1}\boldsymbol{\rho}_{\mathcal{A}}\,. \end{aligned}$$

Lemma 6 *If* $\mathbf{Q}$ *is symmetric decomposable, then* $\mathbf{Q}^{-1}$ *is tridiagonal. In particular, if* $\mathbf{Q} = SDM(\boldsymbol{\alpha}, \boldsymbol{\nu})$, *then* $\mathbf{Q}^{-1} = (\theta_{ij})$ *where* $\theta_{ii} = 1/(\alpha_i - \alpha_{i-1})v_i^2 + 1/(\alpha_{i+1} - \alpha_i)v_i^2$, $\theta_{i,i+1} = \theta_{i+1,i} = -1/(\alpha_{i+1} - \alpha_i)v_i v_{i+1}$ *for* $1 \le i \le n-1$, $\theta_{nn} = 1/(\alpha_n - \alpha_{n-1})v_n^2$, *and* $\theta_{ij} = 0$ *otherwise.*

By Lemma 5 and Lemma 6, $\mathcal{ECWS}$ can be solved in $O(n)$ time. To solve $\mathcal{WS}$, in practice, the active set method takes less than n iterations and hence the total runtime is $O(n^2)$. Note that unlike previous works, the runtime of this convex quadratic programming approach is independent of the wire length L.

Applications

The wire sizing technique is commonly applied to minimize the wire delay and hence to improve the performance of VLSI circuits. As there are typically millions of wires in modern VLSI circuits, and each wire may be sized many many times in order to explore different architecture, logic design and layout during the design process, it is very important for wire sizing algorithms to be very efficient.

Another popular technique for delay minimization of slow signals is to insert buffers (or called repeaters) to strengthen and accelerate the signals. The work of Chu and Wong can be naturally extended to simultaneously handle buffer insertion. It is shown in [4] that the delay minimization problem for a wire by simultaneous buffer insertion and wire sizing can also be formulated as a convex quadratic program and be solved by active set method. The runtime is only m times more than that of wire sizing, where m is the number of buffers inserted. m is typically 5 or less in practice.

About one third of all nets in a typical VLSI circuit are multi-pin nets (i. e., nets with a tree structure to deliver a signal from a source to several sinks). It is important to minimize the delay of multi-pin nets. The work of Chu and Wong can also be applied to optimize multi-pin nets. The extension is described in Mo and Chu [14]. The idea is to integrate the quadratic programming approach into a dynamic programming framework. Each branch of the net is solved as a convex quadratic program while the overall tree structure is handled by dynamic programming.

Open Problems

After more than a decade of active research, the wire sizing problem by itself is now considered a well-solved problem. Some important solutions are [1,2,3,4,5,6,7,8,9,10,11,13,14,15]. The major remaining challenge is to simultaneously apply wire sizing with other interconnect optimization techniques to improve circuit performance. Wire sizing, buffer insertion and gate sizing are three most commonly used interconnect optimization techniques. It has been demonstrated that better performance can be achieved by simultaneously applying these three techniques than applying them sequentially. One very practical problem is to perform simultaneous wire sizing, buffer insertion and gate sizing to a combinational circuit such that the delay of all input-to-output paths are less than a given target and the total wire/buffer/gate resource usage is minimized.

Cross References

- ► Circuit Retiming
- ► Circuit Retiming: An Incremental Approach
- ► Gate Sizing

Recommended Reading

1. Chen, C.-P., Chen, Y.-P., Wong, D.F.: Optimal wire-sizing formula under the Elmore delay model. In: Proc. ACM/IEEE Design Automation Conf., pp. 487–490 ACM, New York (1996)
2. Chen, C.-P., Wong, D.F.: A fast algorithm for optimal wire-sizing under Elmore delay model. In: Proc. IEEE ISCAS, vol. 4, pp. 412–415 IEEE Press, Piscataway (1996)
3. Chen, C.-P., Wong, D.F.: Optimal wire-sizing function with fringing capacitance consideration. In: Proc. ACM/IEEE Design Automation Conf., pp. 604–607 ACM, New York (1997)
4. Chu, C.C.N., Wong, D.F.: Greedy wire-sizing is linear time. IEEE Trans. Comput. Des. **18**(4), 398–405 (1999)
5. Chu, C.C.N., Wong, D.F.: A quadratic programming approach to simultaneous buffer insertion/sizing and wire sizing. IEEE Trans. Comput. Des. **18**(6), 787–798 (1999)
6. Cong, J., He, L.: Optimal wiresizing for interconnects with multiple sources. ACM Trans. Des. Autom. Electron. Syst. **1**(4) 568–574 (1996)
7. Cong, J., Leung, K.-S.: Optimal wiresizing under the distributed Elmore delay model. IEEE Trans. Comput. Des. **14**(3), 321–336 (1995)
8. Fishburn., J.P.: Shaping a VLSI wire to minimize Elmore delay. In: Proc. European Design and Test Conference pp. 244–251. IEEE Compute Society, Washington D.C. (1997)
9. Fishburn, J.P., Schevon, C.A.: Shaping a distributed-RC line to minimize Elmore delay. IEEE Trans. Circuits Syst.-I: Fundam. Theory Appl. **42**(12), 1020–1022 (1995)
10. Gao, Y., Wong, D.F.: Wire-sizing for delay minimization and ringing control using transmission line model. In: Proc. Conf. on Design Automation and Test in Europe, pp. 512–516. ACM, New York (2000)
11. Kay, R., Bucheuv, G., Pileggi, L.: EWA: Efficient Wire-Sizing Algorithm. In: Proc. Intl. Symp. on Physical Design, pp. 178–185. ACM, New York (1997)
12. Kozlov, M.K., Tarasov, S.P., Khachiyan, L.G.: Polynomial solvability of convex quadratic programming. Sov. Math. Dokl. **20**, 1108–1111 (1979)
13. Lillis, J., Cheng, C.-K., Lin, T.-T.: Optimal and efficient buffer insertion and wire sizing. In: Proc. of Custom Integrated Circuits Conf., pp. 259–262. IEEE Press, Piscataway (1995)

14. Mo, Y.-Y., Chu, C.: A hybrid dynamic/quadratic programming algorithm for interconnect tree optimization. IEEE Trans. Comput. Des. **20**(5), 680–686 (2001)
15. Sapatnekar, S.S.: RC interconnect optimization under the Elmore delay model. In: Proc. ACM/IEEE Design Automation Conf., pp. 387–391. ACM, New York (1994)

Work-Function Algorithm for k Servers

1994; Koutsoupias, Papadimitriou

MAREK CHROBAK
Department of Computer Science at Riverside,
University of California at Riverside,
Riverside, CA, USA

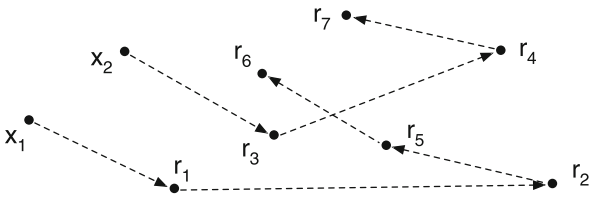
Problem Definition

In the *k-server problem*, the task is to schedule the movement of k servers in a metric space $\mathbb{M}$ in response to a sequence $\varrho = r_1, r_2, \ldots, r_n$ of *requests*, where $r_i \in \mathbb{M}$ for all i. The servers initially occupy some configuration $X_0 \subseteq \mathbb{M}$. After each request r_i is issued, one of the k servers must move to r_i. A *schedule* S specifies which server moves to each request. The task is to compute a schedule with minimum *cost*, where the cost of a schedule is defined as the total distance traveled by the servers. The example below shows a schedule for 2 servers on a sequence of requests.

In the offline case, given $\mathbb{M}$, X_0, and the complete request sequence ϱ, the optimal schedule can be computed in polynomial time [6].

In the *online* version of the problem the decision as to which server to move to each request r_i must be made before the next request r_{i+1} is issued. It is quite easy to see that in this online scenario it is not possible to guarantee an optimal schedule. The accuracy of online algorithms is often measured using competitive analysis. Denote by $cost_{\mathcal{A}}(\varrho)$ the cost of the schedule produced by an online k-server algorithm $\mathcal{A}$ on a request sequence ϱ, and let $opt(\varrho)$ be the cost of an optimal schedule on ϱ. $\mathcal{A}$ is called *R-competitive* if $cost_{\mathcal{A}}(\varrho) \le R \cdot opt(\varrho) + B$, where B is a constant that may depend on $\mathbb{M}$ and X_0. The smallest such R is called the *competitive ratio* of $\mathcal{A}$. Of course, the smaller the R the better.

The k-server problem was introduced by Manasse, McGeoch, and Sleator [13,14], who proved that no (deterministic) on-line algorithm can achieve a competitive ratio smaller than k, in any metric space with at least $k + 1$ points. They also gave a 2-competitive algorithm for $k = 2$ and stated what is now known as the *k-server conjecture*,



Work-Function Algorithm for k Servers, Figure 1
A schedule for 2 servers on a request sequence $\varrho = r_1, r_2, \ldots, r_7$. The initial configuration is $X_0 = \{x_1, x_2\}$. Server 1 serves r_1, r_2, r_5, r_6, while server 2 serves r_3, r_4, r_7. The cost of this schedule is $d(x_1, r_1) + d(r_1, r_2) + d(r_2, r_5) + d(r_5, r_6) + d(x_2, r_3) + d(r_3, r_4) + d(r_4, r_7)$, where $d(x, y)$ denotes the distance between points x, y

which postulates that there exists a k-competitive online algorithm for all k. Koutsoupias and Papadimitriou [10,11] (see also [3,8,9]) proved that the *work-function algorithm* presented in the next section has competitive ratio at most $2k - 1$, which to date remains the best upper bound known.

Key Results

The idea of the work-function algorithm is to balance two greedy strategies when a new request is issued. The first one is to simply serve the request with the closest server. The second strategy attempts to follow the optimum schedule. Roughly, from among the k possible new configurations, this strategy chooses the one where the optimum schedule would be at this time, if no more requests remained to be issued.

To formalize this idea, for each request sequence ϱ and a k-server configuration X, let $\omega_\varrho(X)$ be the minimum cost of serving ϱ under the constraint that at the end the server configuration is X. (Assume that the initial configuration X_0 is fixed.) The function $\omega_\varrho(\cdot)$ is called the *work function* after the request sequence ϱ.

Algorithm WFA

Denote by σ the sequence of past requests, and suppose that the current server configuration is $S = \{s_1, s_2, \ldots, s_k\}$, where s_j is the location of the jth server. Let r be the new request. Choose $s_j \in S$ that minimizes the quantity $\omega_{\sigma r}(S - \{s_j\} \cup \{r\}) + d(s_j, r)$, and move server j to r.

Theorem 1 ([10,11]) *Algorithm* WFA *is* $(2k - 1)$*-competitive.*

Applications

The k-server problem can be viewed as an abstraction of online problems that arise in emergency crew schedul-

ing, caching (or paging) in two-level memory systems, scheduling of disk heads, and other. Nevertheless, in its pure abstract form, it is mostly of theoretical interest.

Algorithm WFA can be applied to some generalizations of the k-server problem. In particular, it is $(2n-1)$-competitive for n-state metrical task systems, matching the lower bound [3,4,8]. See [1,3,5] for other applications and extensions.

Open Problems

Theorem 1 comes tantalizingly close to settling the k-server conjecture described earlier in this section. In fact, it has been even conjectured that Algorithm WFA itself is k-competitive for k servers, but the proof of this conjecture, so far, remains elusive.

For $k \geq 3$, k-competitive online k-server algorithms are known only for some restricted metric spaces, including trees [7], metric spaces with up to $k+2$ points, and the Manhattan plane for $k = 3$ (see [2,6,12]). As the analysis of Algorithm WFA in the general case appears difficult, it would of interest to prove its k-competitiveness for some natural special cases, for example in the plane (with any reasonable metric) for $k \geq 4$ servers.

Very little is known about the competitive ratio of the k-server problem in the randomized case. In fact, it is not even known whether a ratio better than 2 can be achieved for $k = 2$.

Cross References

- ► Algorithm DC-Tree for k Servers on Trees
- ► Deterministic Searching on the Line
- ► Generalized Two-Server Problem
- ► Metrical Task Systems
- ► Online Paging and Caching
- ► Paging

Recommended Reading

1. Anderson, E.J., Hildrum, K., Karlin, A.R., Rasala, A., Saks, M.: On list update and work function algorithms. Theor. Comput. Sci. **287**, 393–418 (2002)
2. Bein, W., Chrobak, M., Larmore, L.L.: The 3-server problem in the plane. Theor. Comput. Sci. **287**, 387–391 (2002)
3. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
4. Borodin, A., Linial, N., Saks, M.: An optimal online algorithm for metrical task systems. In: Proc. 19th Symp. Theory of Computing (STOC), ACM, pp. 373–382 (1987)
5. Burley, W.R.: Traversing layered graphs using the work function algorithm. J. Algorithms **20**, 479–511 (1996)
6. Chrobak, M., Karloff, H., Payne, T.H., Vishwanathan, S.: New results on server problems. SIAM J. Discret. Math. **4**, 172–181 (1991)
7. Chrobak, M., Larmore, L.L.: An optimal online algorithm for k servers on trees. SIAM J. Comput. **20**, 144–148 (1991)
8. Chrobak, M., Larmore, L.L.: Metrical task systems, the server problem, and the work function algorithm. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms: The State of the Art, pp. 74–94. Springer, London (1998)
9. Koutsoupias, E.: Weak adversaries for the k-server problem. In: Proc. 40th Symp. Foundations of Computer Science (FOCS), IEEE, pp. 444–449 (1999)
10. Koutsoupias, E., Papadimitriou, C.: On the k-server conjecture. In: Proc. 26th Symp. Theory of Computing (STOC), ACM, pp. 507–511 (1994)
11. Koutsoupias, E., Papadimitriou, C.: On the k-server conjecture. J. ACM **42**, 971–983 (1995)
12. Koutsoupias, E., Papadimitriou, C.: The 2-evader problem. Inf. Proc. Lett. **57**, 249–252 (1996)
13. Manasse, M., McGeoch, L.A., Sleator, D.: Competitive algorithms for online problems. In: Proc. 20th Symp. Theory of Computing (STOC), ACM, pp. 322–333 (1988)
14. Manasse, M., McGeoch, L.A., Sleator, D.: Competitive algorithms for server problems. J. Algorithms **11**, 208–230 (1990)