P

P2P

2001; Stoica, Morris, Karger, Kaashoek, Balakrishnan

DAHLIA MALKHI
Microsoft, Silicon Valley Campus,
Mountain View, CA, USA

Keywords and Synonyms

Peer to peer; Overlay; Overlay network; DHT; Distributed hash table; CDN; Content delivery network; File sharing; Resource sharing

Problem Definition

This problem is concerned with efficiently designing a serverless infrastructure for a federation of hosts to store, index and locate information, and for efficient data dissemination among the hosts. The key services of peer-to-peer (P2P) overlay networks are:

1. A keyed lookup protocol locates information at the server(s) that hold it.
2. Data store, update and retrieve operations maintain a distributed persistent data repository.
3. Broadcast and multicast support information dissemination to multiple recipients.

Because of their symmetric, serverless nature, these networks are termed *P2P* networks. Below, we often refer to hosts participating in the network as *peers*.

The most influential mechanism in this area is *consistent hashing*, pioneered in a paper by Karger et al. [21]. The idea is roughly the following. Frequently, a good way of arranging a lookup directory is a hash table, giving a fast $O(1)$-complexity data access. In order to scale and provide highly available lookup services, we partition the hash table and assign different chunks to different servers. So, for example, if the hash table has entries 1 through n, and there are k participating servers, we can have each server select a virtual identifier from 1 to n at random. Server i will then be responsible for key values that are closer to i than to any other server identifier. With a good randomization of the hash keys, we can have a more or less balanced distribution of information between our k servers. In expectation, each server will be responsible for (n/k) keys. Furthermore, the departure/arrival of a server perturbs only one or two other servers with adjacent virtual identifiers.

A network of servers that implement consistent hashing is called a *distributed hash table* (DHT). Many current-generation resource sharing networks, and virtually all academic research projects in the area, are built around a DHT idea.

The challenge in maintaining DHTs is two-fold:

Overlay routing Given a hash key i, and starting from any node r in the network, the problem is to find the server s whose key range contains i. The key name i bears no relation to any real network address, such as the IP address of a node, and therefore we cannot use the underlying IP infrastructure to locate s. An overlay routing network links the nodes, and provides them with a routing protocol, such that r can route toward s using the routing target i.

Dynamic maintenance DHTs must work in a highly dynamic environment in which the size of the network is not known a priori, and where there are no permanent servers for maintaining either the hash function or the overlay network (all servers are assumed to be ephemeral). This is especially acute in P2P settings, where the servers are transient users who may come and go as they wish. Hence, there must be a decentralized protocol, executed by joining and leaving peers, that incrementally maintains the structure of the system. Additionally, a joining peer should be able to correctly execute this protocol while initially only having knowledge of a single, arbitrary participating network node.

One of the first overlay network projects was **Chord** [35], after which this encyclopedia entry is named (2001; Sto-

ica, Morris, Karger, Kaashoek, Balakrishnan). More details about Chord are given below.

Key Results

The P2P area is very dynamic and rapidly evolving. The current entry provides a mere snapshot, covering dominant and characteristic strategies, but not offering an exhaustive survey.

Unstructured Overlays

Many of the currently deployed widespread resource-sharing networks have little or no particular overlay structure. More specifically, early systems such as Gnutella version 0.4 had no overlay structure at all, and allowed every node to connect to other nodes arbitrarily. This resulted in severe load and congestion problems.

Two-tier networks were introduced to reduce communication overhead and solve the scalability issues that early networks like Gnutella version 0.4 had. Two-tier networks consist of one tier of relatively stable and powerful nodes, called servers (superpeers, ultrapeers), and a larger tier of clients that search the network though servers. Most current networks, including Edonkey/Emule, KaZaa, and Gnutella, are built using two tiers. Servers provide directory store and search facilities. Searching is either limited to servers to which clients directly connect (eDonkey/eMule) or done by limited-depth flooding among the servers (Gnutella). The two-tier design considerably enhances the scalability and reliability of P2P networks. Nevertheless, the connections among servers and between clients/servers is done in a completely ad hoc manner. Thus, these networks provide no guarantee for the success of searches, nor a bound on their costs.

Structured Overlays Without Locality Awareness

Chord The Chord system was built at MIT and is currently being developed under FNSF's **IRIS** project (http://project-iris.net/). Several aspects of the Chord [35] design have influenced subsequent systems. We briefly explain the core structure of Chord here. Nodes have binary identifiers, assigned uniformly at random. Nodes are arranged in a linked ring according to their virtual identifiers. In addition, each node has shortcut links to other nodes along the ring, link i to a node 2^i away in the virtual identifier space. In this way, one can move gradually to the target by decreasing the distance by half at every step. Routing takes on average $\log n$ hops to reach any target, in a network containing n nodes. Each node maintains approximately $\log n$ links, providing the ability to route to geometrically increasing distances.

Constant Per-Node State Several overlay network algorithms were developed with the goal of pushing the amount of network state kept by each node in the overlay to a minimum. We refer to the state kept by a node as its *degree*, as it mostly reflects the number of connections to other nodes. **Viceroy** [23] was the first to demonstrate a dynamic network in which each node stores only five links to other network nodes, and routes to any other node in a logarithmic number of hops, $\log n$ for a network of n nodes. Viceroy provided a dynamic emulation of a butterfly network (see [11] for a textbook exposition of interconnect networks like butterfly). Later, several emulations of De Bruijn networks emerged, including the generic one of Abraham et al. (AAABMP) [1], the **distance halving** network [26], **D2B** [13], and **Koorde** [20]. Constant-degree overlay networks are too fragile for practical purposes, and may easily degrade in performance or even partition in the face of failures. A study of overlay networks under churn demonstrated these points [18]. Indeed, to the best of our knowledge, none of these constant-degree networks were built. Their main contribution, and the main reason for mentioning these works here, is to know that it is possible in principle to bring the per-node state to a bare, small constant.

Content Addressable Network The Content Addressable Network (CAN) [31] developed at ICSI builds the network as virtual d-dimensional space, giving every node a d-dimensional identifier. The routing topology resembles a d-dimensional torus. Routing is done by following the Euclidean coordinates in every dimension, yielding a $dn^{1/d}$ hop routing strategy. The parameter d can be tuned by the network administrator. Note that for $d = \log n$, CAN's features are the same as in Chord, namely, logarithmic degree and logarithmic routing hop count.

Overlay Routing Inspired by "Small-World" Networks The **Symphony** [24] algorithm emulates routing in a small world. Nodes have k links to nodes whose virtual identifiers are chosen at random according to a routable small-world distribution [22]. With k links, Symphony is expected to find a target in $\log^2 / k$ hops.

Overlay Networks Supporting Range Queries One of the deficiencies of DHTs is that they support only exact key lookup; hence, they do not address well the need to locate a range of keys, or to have a fuzzy search, e. g., search for any key that matches some prefix. **SkipGraphs** [4] and the **SkipNet** [19] scheme from Microsoft (project Herald) independently developed a similar DHT based on a ran-

P2P, Table 1
Comparison of various measures of lookup schemes with no locality awareness

Overlay lookup scheme	Topology resemblance	Hops	Degree
Chord	Hypercube	$\log n$	$\log n$
Viceroy	Butterfly	$\log n$	5
AAABMP, Distance-halving, Koorde, D2B	De Bruijn	$\log n$	4
Symphony	Small world	$\log^2 n/k$	k
SkipGraphs/SkipNet	Skip list	$\log n$	$\log n$
CAN	Torus	$dn^{1/d}$	d

domized skip list [28] that supports range queries over a distributed network. The approach in both of these networks is to link objects into a double-linked list, sorted by object names, over which "shortcut" pointers are built. Pointers from each object skip to a geometric sequence of distances in the sorted list, i. e., the first pointer jumps two items away, the second four items, and so on, up to pointer $\log n - 1$, which jumps over half of the list. Logarithmic, load-balanced lookup is achieved in this scheme in the same manner as in Chord. Because the identifier space is sorted by object names, rather than hash identifiers, ranges of objects can be scanned efficiently simply by routing to the lowest value in the range; the remaining range nodes reside contiguously along the ring.

By prefixing organization names to object names, SkipNet achieves contiguity of nodes belonging to a single organization along the ring, and the ability to map objects on nodes in their local organizations. In this way SkipNet achieves resource proximity and isolation the only system besides RP [33] to have this feature.

Whereas the SkipGraphs work focuses on randomized load-balancing strategies and proofs, the SkipNet system considers issues of dynamic maintenance, variable base sizes, and adopts the locality-awareness strategy of Pastry [33], which is described below.

Summary of Non-Locality-Aware Networks Each of the networks mentioned above is distinct in one or more of the following properties: The (intuitive) emulated **topology**; the expected number of **hops** required to reach a target; and the per-node **degree**. Table 1 summarizes these properties.

Locality Awareness

The problem with the approaches listed above is that they ignore the proximity of nodes in the underlying networks, and allow hopping back and forth across large physical distances in search of content. Recent studies of scalable content exchange networks [17] have indicated that up to 80% of Internet searches could be satisfied by local hosts within one's own organization. Therefore, even one far hop might be too costly. The next systems we encounter consider proximity relations among nodes in order to obtain locality awareness, i. e., that lookup costs are proportional to the actual distance of interacting parties.

Growth-Bounded Networks Several locality-aware lookup networks were built around a bit-fixing protocol that borrows from the seminal work of Plaxton et al. [27] (PRR). The *growth bounded* network model for which this scheme is aimed views the network as a metric space, and assumes that the densities of nodes in different parts of the network are not terribly different. The PRR [27] lookup scheme uses prefix routing, similar to Chord. It differs from Chord in that a link for flipping the ith identifier bit connects with any node whose length-i prefix matches the next hop. In this way, the scheme favors the closest one in the network. This strategy builds *geometric routing*, whose characteristic is that the routing steps toward a target increase geometrically in distance. This is achieved by having large flexibility in the choice of links for each prefix at the beginning of a route, and narrowing it down as the route progresses. The result is an overlay routing scheme that finds any target with a cost that is proportional to the shortest-distance route.

The systems that adopt the PRR algorithm are **Pastry** [33], **Tapestry** [36], and **Bamboo** [32]. A very close variant is **Kademlia** [25], in which links are symmetric. It is worth mentioning that the LAND scheme [2] improves PRR in providing a nearly optimal guaranteed locality guarantee; however, LAND has not been deployed.

Applications

Caching

The Coral network [14] from NYU, built on top of DSHT [15], has been operational since around 2004. It provides free content delivery services on top of the PlanetLab-distributed test bed [9], similar to the commercial services offered by the Akamai network. People use it to provide multiple, fast access points to content they wish to publish on the Web.

Coral optimizes access locality and download rate using locality-aware lookup provided by DSHT. Within Coral, DSHT is utilized to support locality-aware object location in two applications. First, Coral contains a collection of HTTP proxies that serve as content providers;

DSHT is used by clients for locating a close-by proxy. Second, proxy servers themselves use DSHT to locate a nearby copy of content requested by the client, thus making use of copies of the content that are stored in the network, rather than going to the source of the content.

Multicast

Several works deploy an event notification or publish–subscribe service over an existing routing overlay by building reverse-routing multicast paths from a single "target" to all "sources." For example, multicast systems built in this way include the Bayeux network [38], which is built over Tapestry [36], and SCRIBE [5], which is built over Pastry. In order to publish a file, the source advertises using flooding a tuple which contains the semantic name of a multicast session and a unique ID. This tuple is hashed to obtain a node identifier which becomes the session root node. Each node can join this multicast session by sending a message to the root. Nodes along the way maintain membership information, so that a multicast tree is formed in the reverse direction. The file content (and any updates) is flooded down the tree. Narada [8] is built with the same general architecture, but differs in its choice of links, and the maintenance of data.

Routing Infrastructure

A DHT can serve well to store routing and (potentially dynamic) location information of virtual host names. This idea has been utilized in a number of projects. A naming system for the Internet called CoDoNS [30] was built at Cornell University over the BeeHive overlay [29]. CoDoNS provides a safety net and is a possible replacement for the Domain Name System, the current service for looking up host names. Support for virtual IPv6 network addresses is provided in [37] by mapping names to their up-to-date, reachable IPv4 address. The Internet Indirection Infrastructure [34] built at the University of California, Berkeley provides support for virtual Internet host addresses that allows mobility.

Collaborative Content Delivery

Recent advances provide collaborative content delivery solutions that address both load balance and resilience via striping. The content is split into pieces (quite possibly with some redundancy through error-correcting codes). The source pushes the pieces of the file to an initial group of nodes, each of which becomes a source of a distribution tree for its piece, and pushes it to all other nodes. These works demonstrate clearly the advantages of data striping, i. e., of simultaneously exchanging stripes of data, over a tree-based dissemination of the full content.

SplitStream [6] employs the Pastry routing overlay in order to construct multiple trees, such that each participating node is an inner node in only one tree. It then supports parallel download of stripes within all trees. SplitStream [6] strives to obtain load balancing between multicast nodes. It achieves that by splitting the published content into several parts, called stripes, and publishing each part separately. Each stripe is published using a tree-based multicast. The workload is divided between the participating nodes by sending each stripe using a different multicast tree. Load balance is achieved by carefully choosing the multicast trees so that each node serves as an interior node in at most one tree. This reduces the number of "free riders" who only receive data.

A very popular file-distribution network is the BitTorrent system [10]. Nodes in BitTorrent are divided into *seed* nodes and *clients*. Seed nodes contain the desired content in full (either by being original providers, or by having completed a recent download of the content). Client nodes connect with a seed node or several seed nodes, as well as a *tracker* node, whose goal is to keep track of currently downloading clients. Each client selects a group (currently, of size about 20) of other downloading clients, and exchanges chunks of data obtained from the seed(s) with them. BitTorrent employs several intricate strategies for selecting which chunks to request from what other clients, in order to obtain fair load sharing of the content distribution and, at the same time, achieve fast download.

BitTorrent currently does not contain P2P-searching facilities. It relies on central sites known as "trackers" to locate content, and to coordinate the BitTorrent download process. Recent announcements by Bram Cohen (the creator of BitTorrent) and creators of other BitTorrent clients state that new protocols based on BitTorrent will be available soon, in which the role of trackers is eliminated, and searching and coordination is done in a completely P2P manner.

Experience with BitTorrent and similar systems indicates that the main problem with this approach is that towards the end of a download, many peers may be missing the same rare chunks, and the download slows down. Fairly sophisticated approaches were published in an attempt to overcome this issue.

Recently, a number of works at Microsoft Research have demonstrated the benefits of network coding in efficient multicast, e. g., [7] and Avalanche [16]. We do not cover these techniques in detail here, but only briefly state the principal ideas that underlie them.

The basic approach in network coding is to re-encode all the chunks belonging to the file, so that each one that is shared is actually a linear combination of all the pieces. The blocks are then distributed with a description of the content. Once a node obtains these re-encoded chunks, it can generate new combinations from the ones it has, and can send those out to other peers. The main benefit is that peers can make use of any new piece, instead of having to wait for specific chunks that are missing. This means no one peer can become a bottleneck, since no piece is more important than any other. Once a peer collects sufficiently many such chunks, it may use them to reconstruct the whole file.

It is worth noting that in unstructured settings, it was recently shown that network coding offers no advantage [12].

Cross References

- ► Geometric Spanners
- ► Routing
- ► Sparse Graph Spanners

Recommended Reading

1. Abraham, I., Awerbuch, B., Azar, Y., Bartal, Y., Malkhi, D., Pavlov, E.: A generic scheme for building overlay networks in adversarial scenarios. In: Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2003), 2003
2. Abraham, I., Malkhi, D., Dobzinski, O.: LAND: Stretch $(1 + \varepsilon)$ locality aware networks for DHTs. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA04), 2004
3. Abraham, I., Badola, A., Bickson, D., Malkhi, D., Maloo, S., Ron, S.: Practical locality-awareness for large scale information sharing. In: The 4th Annual International Workshop on Peer-To-Peer Systems (IPTPS '05), 2005
4. Aspnes, J., Shah, G.: Skip graphs. In: Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, January 2003, pp. 384–393
5. Castro, M., Druschel, P., Rowstron, A.: Scribe: A large-scale and decentralised application-level multicast infrastructure, IEEE J. Sel. Areas Commun. (JSAC) (Special issue on Network Support for Multicast Communications) **20**(8), 1489–1499 (2002). ISSN: 0733–8716
6. Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron, A., Singh, A.: Splitstream: High-bandwidth multicast in a cooperative environment. In: SOSP'03, October 2003
7. Chou, P., Wu, Y., Jain, K.: Network coding for the internet. In: IEEE Communication Theory Workshop, 2004
8. Chu, Y., Rao, S.G., Zhang, H.: A case for end system multicast. In: Proceedings of ACM SIGMETRICS, Santa Clara, June 2000, pp. 1–12
9. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: Planetlab: An overlay testbed for broad-coverage services. ACM SIGCOMM Comput. Commun. Rev. **33**, 3–12 (2003)
10. Cohen, B.: Incentives build robustness in bittorrent. In: Proceedings of P2P Economics Workshop, 2003
11. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to algorithms. MIT Press (1990)
12. Fernandess, Y., Malkhi, D.: On collaborative content distribution using multi-message gossip. In: Twentieth IEEE International Parallel and Distributed Processing Symposium (IPDPS 2006), Greece, April 2006
13. Fraigniaud, P., Gauron, P.: The content-addressable network D2B. Tech. Report 1349, LRI, Univ. Paris-Sud (2003)
14. Freedman, M.J., Freudenthal, E., Mazières, D.: Democratizing content publication with coral. In: Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04), March 2004
15. Freedman, M.J., Mazières, D.: Sloppy hashing and self-organizing clusters. In: Proceedings of the 2nd Intl. Workshop on Peer-to-Peer Systems (IPTPS '03), February 2003
16. Gkantsidis, C., Rodriguez, P.: Network coding for large scale content distribution. In: IEEE/INFOCOM, 2005
17. Gummadi, K.P., Dunn, R.J., Saroiu, S., Gribble, S.D., Levy, H.M., Zahorjan, J.: Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In: Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 314–329. ACM Press (2003)
18. Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., Stoica, I.: The impact of DHT routing geometry on resilience and proximity. In: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 381–394. ACM Press (2003)
19. Harvey, N.J.A., Jones, M.B., Saroiu, S., Theimer, M., Wolman, A.: Skipnet: A scalable overlay network with practical locality properties. In: Proceedings of Fourth USENIX Symposium on Internet Technologies and Systems (USITS '03), March 2003
20. Kaashoek, F., Karger, D.R.: Koorde: A simple degree-optimal hash table. In: 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), 2003
21. Karger, D., Lehman, E., Leighton, F.T., Levine, M., Lewin, D., Panigrahy, R.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC), 1997, pp. 654–663 1997
22. Kleinberg, J.: The small-world phenomenon: An algorithmic perspective. In: Proc. 32nd ACM Symposium on Theory of Computing (STOC 2000), 2000, pp. 163–170
23. Malkhi, D., Naor, M., Ratajczak, D.: Viceroy: A scalable and dynamic emulation of the butterfly. In: Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC '02), 2002, pp. 183–192
24. Manku, G.S., Bawa, M., Raghavan, P.: Symphony: Distributed hashing in a small world. In: Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003) 2003, pp. 127–140
25. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the XOR metric. In: Proc. 1st Intl. Workshop on Peer-to-Peer Systems (IPTPS 2002), 2002, pp. 53–65
26. Naor, M., Wieder, U.: Novel architectures for p2p applications: the continuous-discrete approach. In: The Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '03), 2003
27. Plaxton, C., Rajaraman, R., Richa, A.: Accessing nearby copies of replicated objects in a distributed environment. In: Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 97), 1997, pp. 311–320

28. Pugh, W.: Skip lists: A probabilistic alternative to balanced trees. In: Workshop on Algorithms and Data Structures, 1989, pp. 437–449
29. Ramasubramanian, V., Sirer, E.G.: Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In: Proceedings of Networked System Design and Implementation (NSDI), 2004
30. Ramasubramanian, V., Sirer, E.G.: The design and implementation of a next generation name service for the internet. In: Proceedings of SIGCOMM, 2004
31. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proceedings of the ACM SIGCOMM 2001 Technical Conference, 2001
32. Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling churn in a dht. Tech. Report Technical Report UCB//CSD-03-1299, The University of California, Berkeley, December 2003
33. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001, pp. 329–350
34. Stoica, I., Adkins, D., Zhuang, S., Shenker, S., Surana, S.: Internet Indirection Infrastructure. In: Proceedings of ACM SIGCOMM, pp. 73–88 (2002)
35. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of the SIGCOMM 2001
36. Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.: Tapestry: A resilient global-scale overlay for service deployment. IEEE J. Sel. Areas Commun. (2003)
37. Zhou, L., van Renesse, R., Marsh, M.: Implementing IPv6 as a Peer-to-Peer Overlay Network. In: Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02), pp. 347 (2002)
38. Zhuang, S.Q., Zhao, B.Y., Joseph, A.D., Katz, R.H., Kubiatowicz, J.: Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In: Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001), 2001

Packet Routing

1988; Leighton, Maggs, Rao

Lenore J. Cowen
Department of Computer Science, Tufts University, Medford, MA, USA

Keywords and Synonyms

Store-and-forward routing; Job shop scheduling

Problem Definition

A collection of packets need to be routed from a set of specified sources to a set of specified destinations in an arbitrary network. Leighton, Maggs and Rao [5] looked at a model where this task is divided into two separate tasks: the first is the *path selection* task, where for each specified packet i with source s_i and packet destination t_i, a simple (meaning edges don't repeat) path P_i through the network from s_i to t_i is pre-selected. Packets traverse the network in a *store and forward* manner: each time a packet is forwarded it travels along the next link in the pre-selected path. It is assumed that only one packet can cross each individual link at each given global (synchronous) timestep. Thus, when there is contention for a link, packets awaiting traversal are stored in the local link's queue (special source and sink queues of unbounded size are also defined that store packets at their origins and destinations). Thus, the second task, and the focus of the Leighton, Maggs and Rao result (henceforth called the LMR result) is the *scheduling* task: a determination, when a link's queue is not empty, of which packet gets to traverse the link in the next timestep (where it is assumed to immediately join the link queue for its next hop). The goal is to schedule the packets so that the *maximum* time that it takes any packet to reach its destination is minimized.

There are two parameters of the network together with the pre-selected paths that are clearly relevant. One is the *congestion* c, defined as the maximum number of paths that all use the same link. The other is the *dilation* d, which is simply the length of the longest path that any packet traverses in the network. Clearly each of c and d is a lower-bound on the length of any schedule that routes all the packets to their destinations. It is easy to see that a schedule of length at most cd always exists. In fact, any schedule that never lets a link go idle if there is a packet that can use that link at that timestep is guaranteed to terminate in cd steps, because each packet traverses at most d links, and at any link can be delayed by at most $c - 1$ other packets.

Key Results

The surprising and beautiful result of LMR is as follows:

Theorem ([5]) *For any network G with a pre-specified set of paths P with congestion c and dilation d, there exists a schedule of length $O(c + d)$, where the queue sizes at each edge are always bounded by a constant.*

The original proof of the LMR paper is non-constructive. That is, it uses the Local Lemma [3] to prove the existence of such a schedule, but does not give a way to find it. In his book [10], Scheideler showed that in fact, a $O(c + d)$ schedule exists with edge queue sizes bounded by 2 (and gave a simpler proof of the original LMR result). A subsequent paper of Leighton, Maggs and Richa in 1999 [6] provides a constructive version of the original LMR paper as follows:

Theorem ([6]) *For any network G with a pre-specified set of paths P with congestion c and dilation d, there exists*

a schedule of length $O(c + d)$. Furthermore, such a schedule can be found in $O(p \log^{1+\epsilon} p \log^*(c + d))$ time for any $\epsilon > 0$, where p is the sum of the lengths of the paths taken by the packets and ε is incorporated into the constant hidden by the big-O in the schedule length.

The algorithm in the paper is a randomized one, though the authors claim that it can be derandomized using the method of conditional probabilities. However, even though the algorithm of Leighton, Maggs and Richa is constructive, it is still an offline algorithm: namely, it requires full knowledge of all packets in the network and the precise paths that each will traverse in order to construct the schedule. The original LMR paper also gave a simple randomized *online* algorithm, that, by assigning delays to packets independently and uniformly at random from an appropriate interval, results in a schedule which is much better than greedy schedules, though not as good as the offline constructions.

Theorem ([5]) *There is a simple randomized on-line algorithm for producing, with high probability, a schedule of length $O(c + d \log(Nd))$ using queues of size $O(\log(Nd))$, where c is the congestion, d is the dilation, and N is the number of packets.*

In the special case where it is assumed that all packets follow shortest paths in the network, Meyer, auf der Heide and Vöcking produced a simple randomized online algorithm that produces, with high probability, a schedule of length $O(c + d + \log Nd)$ steps, but queues can be as large as $O(c)$ [7]. For arbitrary paths, the LMR online result was ultimately improved to $O(c + d + \log^{1+\epsilon} N)$ steps, for any $\epsilon > 0$ with high probability, in a series of two papers by Rabani and Tardos [9], and Rabani and Ostrovsky [8]. Online protocols have also been studied in a setting where additional packets are dynamically injected into the network in adversarial settings, see [10] for a survey.

The discussion is briefly returned to the first task, namely to pre-construct the set of paths. Clearly, the goal is to find, for a particular set of packets with pre-specified sources and destinations, a set of paths that minimizes $c + d$. Srinivasan and Teo [12] designed an off-line algorithm that produces a set of paths whose $c + d$ is provably within a constant factor of optimal. Together with the offline LMR result, that gives a constant-factor approximation problem for the offline store-and-forward packet routing problem. Note that the approach of trying to minimize $c + d$ rather than c alone seems crucial; producing schedules within a constant factor of optimal congestion c is hard, and in fact has been shown to be related to the integrality gap for multicommodity flow [1,2].

Applications

Network Emulations

Typically, a guest network G is emulated by a host network H by embedding G into H. Nodes of G are mapped to nodes of H, while edges of G are mapped to paths in H. If P is the set of e paths (each corresponding to an edge in the guest network G), the congestion and dilation can be defined analogously as in the main result for the set of paths P, namely c denotes the maximum number of paths that use any one edge of H, and d is the length of the longest path in P. In addition, the *load* l is defined to be the maximum number of nodes in G that are mapped to a single node of H. Once G is embedded in H, H can emulate G as follows: Each node of H emulates the local computations performed by the l (or fewer) nodes mapped to it in $O(l)$ time. Then for each packet sent along an edge of G, H sends a packet along the corresponding path in the embedding; using the offline LMR result this takes $O(c + d)$ steps. Thus, H can emulate each step of G in $O(c + d + l)$ steps.

Job Shop Scheduling

Consider a scheduling problem with jobs $j_1, \ldots j_r$ and machines $m_1, \ldots, m_s$ for which each job must be performed on a specified sequence of machines (in a specified order). Assume each job spends unit time on each machine, and that no machine has to work on any job more than once (In the language of job-shop scheduling, this is the *non-preemptive, acyclic, job-shop scheduling problem*, with unit jobs). There is a mapping of sequences of machines to paths and jobs to packets so that this becomes an encoding of the main packet routing problem, where if c is now to be the maximum number of jobs that have to be run on any one machine, and d to be the maximum number of different machines that work on any single job, there becomes $O(c)$ congestion and $O(d)$ dilation for the corresponding packet-routing instance. Then the offline LMR result shows that there is a schedule that completes all jobs in $O(c + d)$ steps, where in addition, each job waits at most a constant number of steps in between consecutive machines (and the queue of jobs waiting for any particular machine will always be bounded by a constant). Similar techniques to those developed in the LMR paper have subsequently been applied to more general instances of Job-Shop Scheduling; see [4,11].

Open Problems

The main open problem is whether there is a randomized *online* packet scheduling that matches the offline LMR bound of $O(c + d)$. The bound of [8] is close, but still grows logarithmically with the total number of packets.

For job shop scheduling, it is unknown whether the constant-factor approximation algorithm for the non-preemptive acyclic job-shop scheduling problem with unit length jobs implied by LMR can be improved to a PTAS. It is also unknown whether there is a constant-factor approximation in the case of arbitrary-length jobs.

Recommended Reading

1. Andrews, M., Zhang, L.: Hardness of the Undirected Congestion Minimization Problem. Proceedings of the 37th Annual ACM Symposium on Theory of Computing, pp. 284–293 (2005)
2. Chuzhoy, J., Naor, J.: New Hardness Results for Congestion Minimization and Machine Scheduling. Proceedings of the 36th Annual ACM Symposium on Theory of Computing, pp. 28–34. ACM, New York (2004)
3. Erdös, P., Lovász, L.: Problems and results on 3-chromatic hypergraphs and some related questions. Colloq. Math. Soc. János Bolyai **10**, 609–627 (1975)
4. Goldberg, L.A., Patterson, M., Srinivasan, A., Sweedick, E.: Better Approximation Guarantees for Job-Shop Scheduling. SIAM J. Discret. Math. **14**(1), 67–92 (2001)
5. Leighton, F.T., Maggs, B.M., Rao, S.B.: Packet routing and job-shop scheduling in O(congestion+dilation) steps. Combinatorica **14**(2), 167–180 (1994)
6. Leighton, F.T., Maggs, B.M., Richa, A.W.: Fast algorithms for finding O(congestion+dilation) packet routing schedules. Combinatorica **19**(3), 375–401 (1999)
7. Meyer auf der Heide, F., Vöcking, B.: Shortest-Path Routing in Arbitrary Networks. J. Algorithms **31**(1), 105–131 (1999)
8. Ostrovsky, R., Rabani, Y.: Universal O(congestion + dilation + $\log^{1+\varepsilon} N$) Local Control Packet Switching Algorithm. In: Proceedings of The Twenty-Ninth ACM Symposium on Theory of Computing, pp. 644–653 (1997)
9. Rabani, Y., Tardos, E.: Distributed Packet Switching in Arbitrary Networks. In: the 28th ACM Symposium on Theory of Computing, pp. 366–376 (1996)
10. Scheideler, C.: Universal Routing Strategies for Interconnection Networks. In: Lecture Notes in Computer Science, vol. 1390. Springer (1998)
11. Shmoys, D.B., Stein, C., Wein, J.: Improved Approximation Algorithms for Shop Scheduling Problems. SIAM J. Comput. **23**(3), 617–632 (1994)
12. Srinivasan, A., Teo, C.P.: A Constant-Factor Approximation Algorithm for Packet Routing and Balancing Local vs. Global Criteria. SIAM J. Comput. **30**(6), 2051–2068 (2000)

Packet Switching in Multi-Queue Switches

2004; Azar, Richter; Albers, Schmidt

MARKUS SCHMIDT
Institute for Computer Science, University of Freiburg, Freiburg, Germany

Keywords and Synonyms

Online packet buffering; Online packet routing

Problem Definition

A multi-queue network switch serves m incoming queues by transmitting data packets arriving at m input ports through one single output port. In each time step, an arbitrary number of packets may arrive at the input ports, but only one packet can be passed through the common output port. Each packet is marked with a value indicating its priority in the Quality of Service (QoS) network. Since each queue has bounded capacity B and the rate of arriving packets can be much higher than the transmission rate, packets can be lost due to insufficient queue space. The goal is to maximize the throughput which is defined as the total value of transmitted packets. The problem comprises two dependent questions: buffer management, namely which packets to admit into the queues, and scheduling, i. e. which (FIFO) queue to use for transmission in each time step.

Two scenarios are distinguished: (a) unit packet value (All packets have the same value.), (b) arbitrary packet values.

The problem is considered as an online problem, i. e. at time step t, only the packet arrivals until t are known, but nothing about future packet arrivals. The online switch performance in QoS based networks is studied by using competitive analysis in which the throughput of the online algorithm is compared to the throughput of an optimal offline algorithm knowing the whole arrival sequence in advance.

If not stated otherwise, the admission control is assumed to allows preemption, i. e. packets once enqueued need not necessarily be transmitted, but can be discarded.

Problem 1 (Unit value problem) *All packets have value 1. Since all packets are thus equally important, the admission control policies simplify: All arriving packets are to be enqueued; in the case of buffer overflow, it does not matter which packets are stored in the queue and which packets are discarded.*

Problem 2 (General problem) *Each packet has its individual value where usually a range $[1, \alpha]$ is given for all packets. A special case consists in the two value model where the values are restricted to $\{1, \alpha\}$.*

Key Results

Unit value packets
Deterministic algorithms

Theorem 1 ([1]) *For any buffer size B, the competitive ratio of each deterministic online algorithm is not smaller than $(e_B + \frac{2}{B})/(e_B - 1 + \frac{1}{B}) \geq \frac{e}{e-1} \approx 1.58$ where $e_B = ((B+1)/B)^B$.*

Theorem 2 ([4]) *Every work-conserving online algorithm is 2-competitive.*

Theorem 3 ([1]) *For any buffer size B, the competitive ratio of any greedy algorithm, which always serves a longest queue (LQF), is at least* $2 - \frac{1}{B}$ *if* $m \gg B$.

Algorithm: *SGR* (Semi-Greedy)
In each time step, the algorithm executes the first rule that applies to the current buffer configuration.

1. If there is a queue buffering more than $\lfloor B/2 \rfloor$ packets, serve the queue currently having the maximum load.
2. If there is a queue the hitherto maximum load of which is less than B, serve among these queues the one currently having the maximum load.
3. Serve the queue currently having the maximum load.

Ties are broken by choosing the queue with the smallest index. The hitherto maximum load is reset to 0 for all queues whenever all queues are unpopulated in *SGR*'s configuration.

Theorem 4 ([1]) *If B is even, then SGR is* $\frac{17}{9} \approx 1.89$*-competitive. If B is odd, then SGR is* $(\frac{17}{9} + \frac{\delta_B}{9})$*-competitive where* $\delta_B = \frac{2}{B+1}$.

Theorem 5 ([3]) *Algorithm* $E^{M^{\acute{E}P'}}$ *(not stated in detail due to space limitation), which is based on a water level algorithm and uses a fractional matching in an online constructed graph, achieves a competitiveness of* $e/(e-1)(1 + (\lfloor H_m + 1 \rfloor)/B)$*, where* H_m *denotes the* m^{th} *harmonic number. Thus,* $E^{M^{\acute{E}P'}}$ *is asymptotically* $\frac{e}{e-1}$*-competitive for* $B \gg \log m$.

Randomized algorithms

Theorem 6 ([1]) *The competitive ratio of each randomized online algorithm is at least* $\varrho = 1.4659$ *for any buffer size B (*$\varrho = 1 + \frac{1}{\alpha+1}$ *where* α *is the unique positive root of* $e^{\alpha} = \alpha + 2$*).*

Theorem 7 (Generalizing technique [9]**)** *If there is a randomized c-competitive algorithm A for* $B = 1$*, then there is a randomized c-competitive algorithm* $\tilde{A}$ *for all B.*

Algorithm: *RS* (Random Schedule)

1. The algorithm uses m auxiliary queues $Q_1, \ldots, Q_m$ of sizes $B_1, \ldots, B_m$ (different buffer sizes at the distinct ports are allowed), respectively. These queues contain real numbers from the range (0,1), where each number is labeled as either marked or unmarked. Initially, these queues are empty.
2. Packet arrival: If a new packet arrives at queue q_i, then the algorithm chooses uniformly at random a real number from the range (0,1) that is inserted into queue Q_i and labeled as unmarked. If queue Q_i was full when the packet arrived, the number at the head of the queue is deleted prior to the insertion of the new number.
3. Packet transmission: Check whether queues $Q_1, \ldots, Q_m$ contain any unmarked number. If there are unmarked numbers, let Q_i be the queue containing the largest unmarked number. Change the label of the largest number to "marked" and select queue q_i for transmission. Otherwise (no unmarked number), transmit a packet from any non-empty queue if such exists.

Theorem 8 ([4]) *Randomized algorithm RS is* $\frac{e}{e-1} \approx 1.58$*-competitive.*

Algorithm: *RP* (Random Permutation)
Let $\mathcal{P}$ be the set of permutations of $\{1, \ldots, m\}$, denoted as m-tuples. Choose $\pi \in \mathcal{P}$ according to the uniform distribution and fix it. In each transmission step, choose among the populated queues that one whose index is most to the front in the m-tuple π.

Theorem 9 ([9]) *Randomized algorithm RP is* $\frac{3}{2}$*-competitive for* $B = 1$*. By Theorem 7, there is a randomized algorithm* $\tilde{RP}$ *that is* $\frac{3}{2}$*-competitive for arbitrary B.*

Arbitrary value packets

Definition 1 A switching algorithm *ALG* is called *comparison-based* if it bases its decisions on the relative order between packet values (by performing only comparisons), with no regard to the actual values.

Theorem 10 (Zero-one principle [5]**)** *Let ALG be a comparison-based switching algorithm (deterministic or randomized). ALG is c-competitive if and only if ALG achieves a c-competitiveness for all packet sequences whose values are restricted to* $\{0, 1\}$ *for every possible way of breaking ties between equal values.*

Algorithm: *GR* (Greedy)
Enqueue a new packet if

- the queue is not full
- or a packet with the smallest value in the queue has a lower value then the new packet. In this case, a smallest value packet is discarded and the new packet in enqueued.

Algorithm: *TLH* (Transmit Largest Head)

1. Buffer management: Use algorithm *GR* independently in all m incoming queues.

2. Scheduling: At each time step, transmit the packet with the largest value among all packets at the head of the queues.

Theorem 11 ([5]) *Algorithm TLH is 3-competitive.*

Algorithm: *TL* (Transmit Largest)
1. Buffer management: Use algorithm *GR* independently in all m incoming queues.
2. Scheduling: At each time step, transmit the packet with the largest value among all packets stored in the queues.

Algorithm: GS^A (Generic Switch)
1. Buffer management: Apply buffer management policy A to all m incoming queues.
2. Scheduling: Run a simulation of algorithm *TL* (in the preemptive relaxed model) with the online input sequence σ. Adopt all scheduling decisions of *TL*, i. e. at each time step, transmit the packet at the head of the queue used by *TL* simulation.

Theorem 12 (General reduction [4]) *Let GS^A denote the algorithm obtained by running algorithm GS with the event-driven single-queue buffer management policy A (preemptive or non-preemptive) and let c_A be the competitive ratio of A. The competitive ratio of GS^A satisfies $c_{GS^A} \leq 2 \cdot c_A$.*

Applications

The unit value scenario models most current networks, e. g. IP networks which only support a "best effort" service in which all packet streams are treated equally, whereas the scenario with arbitrary packet values integrates full QoS capabilities.

The general reduction technique allows to restrict oneself to investigate single-queue buffer problems. It can be applied to a 1.75-competitive algorithm named *PG* by Bansal et al. [7], which achieves the best ratio known today, and yields an algorithm GS^{PG} that is 3.5-competitive for multi-queue buffers (3.5 is still higher than 3 which is the competitive ratio of TLH). In the 2-value preemptive model, Lotker and Patt-Shamir [8] presented a *mark&flush* algorithm *mf* that is 1.30-competitive for single queue buffers and that the general reduction technique transforms into a 2.60-competitive algorithm GS^{mf} for multi-queue buffers.

For the general non-preemptive model, Andelman et al. [2] presented a policy for a single queue called *Exponential-Interval Round-Robin* (*EIRR*), which is $(e\lceil \ln \alpha \rceil)$-competitive, and showed also a lower bound of $\Theta(\log \alpha)$. In the multi-queue buffer case, the general reduction technique provides a non-preemptive $(e\lceil \ln \alpha \rceil)$-competitive algorithm.

Open Problems

It is known from Theorem 3 that the competitive ratio of any greedy algorithm in the unit value model is at least 2 if $m \gg B$. Which is the tight upper bound for greedy algorithms in the opposite case $B \gg m$?

The proof of the lower bound $e/(e-1)$ in Theorem 1 uses $m \gg B$ whereas Theorem 5 achieves $e/(e-1)$ as an upper bound for $B \gg \log m$. In [4], a lower bound of 1.366 is shown, independent of B and m. Which is the optimal competitive ratio for arbitrary B and m?

Due to the general reduction technique in Theorem 7, the competitive ratio for multi-queue buffer algorithms can be improved if better competitiveness results for single queue buffer algorithms are achieved. Currently, $\frac{\sqrt{13}+5}{6} \approx 1.43$ [2] and 1.75 [7] are the best known lower and upper bounds, respectively. How to reduce this gap?

Cross References

- ▶ Packet Switching in Single Buffer
- ▶ Paging

Recommended Reading

1. Albers, S., Schmidt, M.: On the performance of greedy algorithms in packet buffering. SIAM J. Comput. **35**, 278–304 (2005)
2. Andelman, N., Mansour, Y., Zhu, A.: Competitive queueing policies for QoS switches. In: Proc. 14th ACM-SIAM Symp. on Discrete Algorithms (SODA), 761–770 (2003)
3. Azar, Y., Litichevskey, M.: Maximizing throughput in multi-queue switches. In: Proc. 12th Annual European Symp. on Algorithms (ESA), 53–64 (2004)
4. Azar, Y., Richter, Y.: Management of multi-queue switches in QoS Networks. In: Proc. 35th ACM Symp. on Theory of Computing (STOC), 82–89 (2003)
5. Azar, Y., Richter, Y.: The zero-one principle for switching networks. In: Proc. 36th ACM Symp. on Theory of Computing (STOC), 64–71 (2004)
6. Azar, Y., Richter, Y.: An improved algorithm for CIOQ switches. In: Proc. 12th Annual European Symp. on Algorithms (ESA). LNCS, vol. 3221, 65–76 (2004)
7. Bansal, N., Fleischer, L., Kimbrel, T., Mahdian, M., Schieber, B., Sviridenko, M.: Further improvements in competitive guarantees for QoS buffering. In: Proc. 31st International Colloquium on Automata, Languages, and Programming (ICALP), 64–71 (2004)
8. Lotker, Z., Patt-Shamir, B.: Nearly optimal FIFO buffer management for two packet classes. Comput. Netw. **42**(4), 481–492 (2003)
9. Schmidt, M.: Packet buffering: randomization beats deterministic algorithms. In: Proc. 22nd Annual Symp. on Theoretical Aspects of Computer Science (STACS). LNCS, vol. 3404, 293–304 (2005)

Packet Switching in Single Buffer

2003; Bansal, Fleischer, Kimbrel, Mahdian, Schieber, Sviridenko

Rob van Stee
Algorithms and Complexity Department, Max Planck Institute for Computer Science, Saarbrücken, Germany

Keywords and Synonyms

Buffering

Problem Definition

In this entry, consider a Quality of Service (QoS) buffering system that is able to hold B packets. Time is slotted. At the beginning of a time step a set of packets (possibly empty) arrives and at the end of the time step a single packet may leave the buffer to be transmitted. Since the buffer has a bounded size, at some point packets may need to be dropped. The buffer management algorithm has to decide at each step which of the packets to drop and which packets to transmit, subject to the buffer capacity constraint. The value of a packet p is denoted by $v(p)$. The system obtains the value of the packets it sends, and gains no value otherwise. The aim of the buffer management algorithm is to maximize the total value of transmitted packets.

In the FIFO model, the packet transmitted at time t is always the first (oldest) packet in the buffer.

In the *nonpreemptive* model, packets accepted to the queue will be transmitted eventually and cannot be dropped. In this model, the best competitive ratio achievable is $\Theta(\log \alpha)$ where α is the ratio of the maximum value of a packet to the minimum [1,2].

In the *preemptive* model, packets can also be dropped at some later time before they are served. The rest of this entry focuses on this model. Mansour, Patt-Shamir, and Lapid [9] were the first to study preemptive queuing policies for a single FIFO buffer, proving that the natural greedy algorithm (see definition in Fig. 1) maintains a competitive ratio of at most 4. This bound was improved to the tight value of 2 by Kesselman, Lotker, Mansour, Patt-Shamir, Schieber, and Sviridenko [6].

The greedy algorithm is not optimal since it never preempts a packet until the buffer is full and this might be too late. The first algorithm with a competitive ratio strictly below 2 was presented by Kesselman, Mansour, and van Stee [7]. This algorithm uses a parameter β and introduces an extra rule for processing arrivals, that is executed before rules 1 and 2 of the greedy algorithm. This rule is formulated in Fig. 2.

The Greedy Algorithm.
When a packet of value $v(p)$ arrives:
1. Accept p if there is free space in the buffer.
2. Otherwise, reject (drop or preempt) the packet p' that has minimal value among p and the packets in the buffer. If $p' \neq p$, accept p.

Packet Switching in Single Buffer, Figure 1
The natural greedy algorithm

0. Preempt (drop) the first packet p' in the FIFO order such that $v(p') \leq v(p)/\beta$, if any (p *preempts* p').

Packet Switching in Single Buffer, Figure 2
Extra rule for the preemptive greedy algorithm

0'. Find the first (i. e., closest to the front of the buffer) packet p' such that p' has value less than $v(p)/\beta$ and not more than the value of the packet after p' in the buffer (if any). If such a packet exists, drop it (p *preempts* p').

Packet Switching in Single Buffer, Figure 3
Modified preemptive greedy

It is shown in [7] that by taking $\beta = 15$, the algorithm preemptive greedy (PG) has a competitive ratio of 1.983. The analysis is rather complicated and is done by assigning the value of packets served by the offline algorithm to packets served by PG.

A lower bound of 5/4 for this problem was shown in [9]. This was improved to $\sqrt{2}$ in [2] and then to 1.419 in [7].

Key Results

A modification of PG was presented by Bansal, Fleischer, Kimbrel, Mahdian, Schieber, and Sviridenko [3]. It changes rule 0 to rule 0'.

Thus, the modification compared to PG is that this algorithm finds a "locally optimal" packet to evict. We will denote modified preemptive greedy by MPG.

Theorem 1 ([3]) *For $\beta = 4$, MPG has a competitive ratio of 1.75.*

The proof begins by showing that in order to analyze the performance of MPG, it is sufficient to consider only input instances in which the value of each packet is either 0 or β^i for some $i \geq 0$, but ties are allowed to be broken by the adversary.

The authors then define an *interval structure* for input instances. An interval I is said to be of type i if at every step $t \in I$ MPG outputs a packet of value at least β^i, and I is a maximal interval with this property.

$\mathcal{I}_i$ is the collection of maximal intervals of type i, and $\mathcal{I}$ is the union of all $\mathcal{I}_i$'s. This is a multiset, since an interval of type i can also be contained in an interval of one or more types $j < i$.

This induces an interval structure which is a sequence of ordered rooted trees in a natural way: the root of each tree is an interval in $\mathcal{I}_0$, and the children of each interval $I \in \mathcal{I}_i$ are all the maximal intervals of type $i + 1$ which are contained in I. These children are ordered from left to right based on time, as are the trees themselves. The intervals of type i (and the vertices that represent them) are distinguished by whether or not an eviction of a packet of value at least β^i occurred during the interval.

To complete the proof, the authors show that for every interval structure $\mathcal{T}$, the competitive ratio of MPG on instances with interval structure $\mathcal{T}$ can be bounded by the solution of a linear program indexed by $\mathcal{T}$. Finally, it is shown that for every $\mathcal{T}$ and every $\beta \geq 4$, the solution of this program is at most $2 - 1/\beta$.

Applications

In recent years, there has been a lot of interest in Quality of Service networks. In regular IP networks, packets are indistinguishable and in case of overload any packet may be dropped. In a commercial environment, it is much more preferable to allow better service to higher-paying customers or customers with critical requirements. The idea of Quality of Service guarantees is that packets are marked with values which indicate their importance.

This naturally leads to decision problems at network switches when many packets arrive and overload occurs. The algorithm presented in this entry can be used to maximize network performance in a network which supports Quality of Service.

Open Problems

Despite substantial advances in improving the upper bound for this problem, a fairly large gap remains. Sgall [5] showed that the performance of PG is as good as that of MPG. Recently, Englert and Westermann [4] showed that PG has a competitive ratio of at most $\sqrt{3} \approx 1.732$ and at least $1 + 1/2\sqrt{2} \approx 1.707$. Thus, to improve further, a different algorithm will be needed.

The authors also note that Lotker and Patt-Shamir [8] studied the special case of two packet values and derived a 1.3-competitive algorithm, which closely matches the corresponding lower bound of 1.28 from Mansour et al. [9]. An open problem is to close the remaining small gap.

Cross References

► Packet Switching in Multi-Queue Switches

Recommended Reading

1. Aiello, W., Mansour, Y., Rajagopolan, S., Rosen, A.: Competitive queue policies for differentiated services. In: Proc. of the IEEE INFOCOM, pp. 431–440. IEEE, Tel-Aviv, Israel (2000)
2. Andelman, N., Mansour, Y., Zhu, A.: Competitive queueing policies in QoS switches. In: Proc. 14th Symp. on Discrete Algorithms (SODA), pp. 761–770 ACM/SIAM, San Francisco, CA, USA (2003)
3. Bansal, N., Fleischer, L., Kimbrel, T., Mahdian, M., Schieber, B., Sviridenko, M.: Further improvements in competitive guarantees for QoS buffering. In: Proc. 31st International Colloquium on Automata, Languages, and Programming (ICALP). Lecture Notes in Computer Science, vol. 3142, pp. 196–207. Springer, Berlin (2004)
4. Englert, M., Westermann, M.: Lower and upper bounds on FIFO buffer management in qos switches. In: Azar, Y., Erlebach, T. (eds.) Algorithms – ESA 2006, 14th Annual European Symposium, Proceedings. Lecture Notes in Computer Science, vol. 4168, pp. 352–363. Springer, Berlin (2006)
5. Jawor, W.: Three dozen papers on online algorithms. SIGACT News **36**(1), 71–85 (2005)
6. Kesselman, A., Lotker, Z., Mansour, Y., Patt-Shamir, B., Schieber, B., Sviridenko, M.: Buffer overflow management in QoS switches. SIAM J. Comput. **33**(3), 563–583 (2004)
7. Kesselman, A., Mansour, Y., van Stee, R.: Improved competitive guarantees for QoS buffering. In: Di Battista, G., Zwick, U. (eds.) Algorithms – ESA 2003, Proceedings Eleventh Annual European Symposium. Lecture Notes in Computer Science, vol. 2380, pp. 361–373. Springer, Berlin (2003)
8. Lotker, Z., Patt-Shamir, B.: Nearly optimal FIFO buffer management for DiffServ. In: Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC 2002), pp. 134–142. ACM, New York (2002)
9. Mansour, Y., Patt-Shamir, B., Lapid, O.: Optimal smoothing schedules for real-time streams. In: Proc. 19th Symp. on Principles of Distributed Computing (PODC), pp. 21–29. ACM, New York (2000)

PAC Learning

1984; Valiant

Joel Ratsaby
Department of Electrical and Electronic Engineering, Ariel University Center of Samaria, Ariel, Israel

Keywords and Synonyms

Probably approximately correct learning

Problem Definition

Valiant's work defines a model for representing the general problem of learning a Boolean concept from examples. The motivation comes from classical fields of artificial intelligence [2], pattern classification [5] and machine learning [10]. Classically, these fields have employed numerous heuristics for representing knowledge and defining criteria by which computer algorithms can learn. The pioneering work of [12,13] provided the leap from heuristic-based approaches to a rigorous statistical theory of pattern recognition (see also [1,4,11]). Their main contribution was the introduction of probabilistic upper bounds on the generalization error which hold uniformly over a whole class of concepts. Valiant's main contribution is in formalizing this probabilistic theory into a general model for computational inference. This model which is known as the *Probably Approximately Correct* (PAC) model of learnability is concerned with computational complexity of learning. In his formulation, learning is depicted as an interaction between a teacher and a learner with two main procedures, one which provides randomly drawn examples x of the concept c that is being learned and the second acts as an oracle which provides the correct classification label $c(x)$. Based on a finite number of such examples drawn identically and independently according to *any* fixed probability distribution, the aim of the learner is to infer an approximation of c which is correct with high confidence. Using the terminology of [9] suppose X denotes the space of instances, i. e., objects which a learner can obtain as training examples. A *concept* over X is a Boolean mapping from X to $\{0, 1\}$. Let $\mathbb{P}$ be any fixed probability distribution over X and c a fixed *target* concept to be learned. For any hypothesis concept h over X define by $L(h) = \mathbb{P}(c(x) \neq h(x))$ the *error* of h, i. e., the probability that h disagrees with c on a test instance x which is drawn according to $\mathbb{P}$. Then according to Valiant, an algorithm $\mathbb{A}$ for learning c is one which runs in time t and with a sample of size m where both t and m are polynomials with respect to some parameters (to be specified below) and produces a hypothesis concept h such that with high confidence $L(h)$ is small.

Key Results

The main result of Valiant's work is a formal definition of what constitutes a *learnable* problem. Formally, this is stated as follows: Let $\mathcal{H}$ be a class of concepts over X. Then $\mathcal{H}$ is *learnable* if there exists an algorithm $\mathbb{A}$ with the following property: for every possible target concept $c \in \mathcal{H}$, for every probability distribution $\mathbb{P}$ on X (this is sometimes referred to as the 'distribution-independence' assumption), for all values of a confidence parameter $0 < \delta < 1/2$ and an approximation accuracy parameter $0 < \epsilon < 1/2$, if $\mathbb{A}$ receives as input the value of δ, ϵ and a sample $S = \{(x_i, c(x_i))\}_{i=1}^{m}$ of cardinality m (which may depend on ε and δ) which consists of examples x_i that are randomly drawn according to $\mathbb{P}$ and labeled by an oracle as $c(x_i)$ then with probability $1 - \delta$, $\mathbb{A}$ outputs a hypothesis concept $h \in \mathcal{H}$ such that the error $L(h) \leq \epsilon$. That ϵ can be arbitrarily close to zero follows from what is known as the 'noise-free' assumption, i. e., that the labels comprise the true value of the target concept. If $\mathbb{A}$ runs in time t and if t and m are polynomial in $1/\epsilon$ and $1/\delta$ then $\mathcal{H}$ is *efficiently* PAC learnable.

Valiant has shown that the following classes are all PAC learnable: class of conjunctive normal form expressions with a bounded number of literals in each clause, the class of monotone disjunctive normal form expressions (here the learner requires in addition to S also an oracle that can answer membership queries, i. e., provide the true label $c(x)$ for an x in question), and the class of arbitrary expressions in which each variable occurs just once (using more powerful oracles). Work following Valiant's paper (see [8] for references) has shown that the classes of k-DNF, k-CNF and k-decision lists are PAC learnable for each fixed k. The class of concepts in the form of a disjunction of two conjunctions is not PAC learnable and neither is the class of existential conjunctive concepts on structural instance spaces with two objects. Linear threshold concepts (perceptrons) are PAC learnable on both Boolean and real-valued instance spaces but the class of concepts in the form of a conjunction of two linear threshold concepts is not PAC learnable. The same holds for disjunctions and linear thresholds of linear thresholds (i. e., multilayer perceptrons with two hidden units). If the weights are restricted to 1 and 0 (but the threshold is arbitrary) then linear threshold concepts on Boolean instances spaces are not PAC learnable.

It should be noted that the notion of PAC learnability discussed throughout this entry is sometimes referred to as "proper" PAC learnability because of the requirement that, when learning a concept class $\mathcal{H}$, the learning algorithm must output a hypothesis that also belongs to $\mathcal{H}$. Several of the negative results mentioned above can be circumvented in a model of "improper" PAC learning, where the learning algorithm is allowed to output hypotheses from a broader class of functions than $\mathcal{H}$. See [9] and the proceedings of the COLT conferences for many results of this type.

Applications

Valiant's paper is a milestone in the history of the area known as *Computational Learning Theory* (see proceed-

ings of COLT conferences). The PAC model has been criticized in that the distribution independence assumption and the notion of target concepts with noise free training data are unrealistic in practice, e. g., in machine learning and AI. There has thus been much work on learning models that relax several of the assumptions in Valiant's PAC model. For instance, models which allow noisy labels or remove the assumptions on the independence of training examples, relax the assumption on the probability distribution to be fixed, allow the bounds to be distribution dependent, permit the training sample to be picked by the learner and labeled by the oracle instead of the random sample, or chosen by a helpful teacher. For references, see Sect. 2.6 of [1]. An important followup of Valiant's model was the work of [3] who unified his model with the uniform convergence results of [13]. They showed the important dependence between the notion of learnability and certain combinatorial properties of concept classes, one of which is known as the Vapnik-Chervonenkis (VC) dimension (see Sect. 3.4 of [1] for history on the VC-dimension).

Cross References

- ▶ Attribute-Efficient Learning
- ▶ Hardness of Proper Learning
- ▶ Learning with the Aid of an Oracle
- ▶ Learning Constant-Depth Circuits
- ▶ Learning DNF Formulas
- ▶ Learning with Malicious Noise

Recommended Readings

For a recommended collection of works on the PAC model and its extensions see [6,7,8].

1. Anthony, M., Bartlett, P.L.: Neural Network Learning: Theoretical Foundations. Cambridge University Press, Cambridge, England (1999)
2. Barr, A., Feigenbaum, E.A.: The Handbook of Artificial Intelligence. Addison-Wesley Pub (Sd) (1994)
3. Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.: Learnability and the Vapnik–Chervonenkis dimension. J. ACM **36**(4), 929–965 (1989)
4. Devroye, L., Gyorfi, L., Lugosi, G.: A Probabilistic Theory of Pattern Recognition. Springer, New York, USA (1996)
5. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley-Interscience Publication (2000)
6. Haussler, D.: Applying valiants learning framework to ai concept learning problems. In: Michalski, R., Kodratoff, Y. (eds.) Machine Learning: An Artificial Intelligence Approach. Morgan Kaufmann
7. Haussler, D.: Decision theoretic generalizations of the PAC model for neural net and other learning applications. Inf. Comput. **100**(1), 78–150 (1992)
8. Haussler, D.: Probably approximately correct learning and decision-theoretic generalizations. In: Smolensky, P., Mozer, M., Rumelhart, D. (eds.) Mathematical Perspectives on Neural Networks, pp. 651–718. L. Erlbaum Associates, Mahwah, New Jersey (1996)
9. Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. M.I.T. Press, London, England (1997)
10. Mitchell, T.: Machine Learning. McGraw Hill (1997)
11. Pearl, J.: Capacity and error-estimates for boolean classifiers with limited complexity. IEEE Trans. on Pattern Recognition and Machine Intelligence, PAMI-**1**(4), 350–356 (1979)
12. Vapnik, V.N.: Estimations of dependences based on statistical data. Springer (1982)
13. Vapnik, V.N., Chervonenkis, A.Y.: On the uniform convergence of relative frequencies of events to their probabilities. Theory Probab. Apl. **16**, 264–280 (1971)

PageRank Algorithm

1998; Brin, Page

MONIKA HENZINGER
Google Switzerland & Ecole Polytechnique Federale de Lausanne (EPFL), Lausanne, Switzerland

Problem Definition

Given a user query current web search services retrieve all web pages that contain the query terms, resulting in a huge number of web pages for the majority of searches. Thus it is crucial to reorder or *rank* the resulting documents with the goal of placing the most relevant documents first. Frequently, ranking uses two types of information: (1) query-specific information and (2) query-independent information. The query-specific part tries to measure how relevant the document is to the query. Since it depends to a large part on the content of the page, it is mostly under the control of the page's author. The query-independent information tries to estimate the quality of the page in general. To achieve an objective measure of page quality it is important that the query-independent information incorporates a measure that is not controlled by the author. Thus the problem is to find a measure of page quality that (a) cannot be easily manipulated by the web page's author and (b) works well for *all* web pages. This is challenging as web pages are extremely heterogeneous.

Key Results

The hyperlink structure of the web is a good source for basing such a measure as it is hard for one author or a small set of authors to influence the whole structure, even though they can manipulate a subset of the web pages. Brin and Page showed that a relatively simple analysis of

the hyperlink structure of the web can be used to produce a quality measure for web documents that leads to large improvements in search quality. The measure is called the *PageRank* measure.

Linear Algebra-based Definition

Let n be the total number of web pages. The PageRank vector is an n-dimensional vector with one dimension for each web page. Let d be a small constant, like 1/8, let $deg(p)$ denote the number of hyperlinks in the body text of page p and let $PR(p)$ denote the PageRank value of page p. Assume first that every page contains at least one hyperlink. In such a collection of web pages the PageRank vector is computed by solving a system of linear equations that contains for each page p the equation

$$PR(p) = d/n + (1-d) * \sum_{q \text{ has hyperlink to } p} PR(q)/deg(q) .$$

In matrix notation the PageRank vector is the Eigenvector with 1-Norm one of the matrix A with $d/n+(1-d)/deg(q)$ for entryA_{qp} if q has a hyperlink to p and d/n otherwise.

If web pages without hyperlinks are allowed in this linear system then they might become "PageRank sinks", i. e., they would "receive" PageRank from the pages pointing to them, but would not "give out" their PageRank, potentially resulting in an "unusually high" PageRank value for themselves. Brin and Page proposed two ways to deal with web pages without out-links, namely either to recursively delete them until no such web pages exist anymore in the collection or to add a hyperlink from each such page to *every* other page.

Random Surfer Model

Let the *web graph* $G = (V, E)$ be a directed graph such that each node corresponds to a web page and every hyperlink corresponds to a directed edge from the referencing node to the referenced node. The PageRank can also be interpreted as the following random walk in the web graph. The random walk starts at a random node in the graph. Assume in step k it visits page q. Then it flips a biased coin and with probability d or if q has no out-edges, it selects a random node out of V and visits it in step $k+1$. Otherwise it selects a random out-edge of the current node and visits it in step $k+1$. (Note that this corresponds to adding a directed edge from every page without hyperlink to *every* node in the graph.) Under certain conditions (which do not necessarily hold on the web) the stationary distribution of this random walk corresponds to the PageRank vector. See [1,4] for details.

Brin and Page also suggested to compute the PageRank vector approximately using the power method, i. e., by setting all initial values to $1/n$ and then repeatedly using the PageRank vector of the previous iteration to compute the PageRank vector of the current iteration using the above linear equations. After a hundred iterations barely any values change and the computation is stopped.

Applications

The PageRank measure is used as one of the factors by Google in its ranking of search results. The PageRank computation can be applied to other domains as well. Two examples are reputation management in P2P networks and learning word dependencies in natural language processing. In relational databases PageRank was used to weigh database tuples in order to improve keyword searching when a user does not know the schema. Finally, in rank aggregation PageRank can be used to find a permutation that minimally violates a set of given orderings. See [1] for more details.

Variations of PageRank were studied as well. Personalizing the PageRank computation such that the values reflect the interest of a user has received a lot of attention. See [3] for a survey on this topic. It can also be modified to be used for detecting web search spam, i. e., web pages that try to manipulate web search results [1].

Recommended Reading

1. Berkhin, P.: A survey on PageRank computing. Internet Math. **2**(1), 73–120 (2005)
2. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. In: Proc. 7th Int. World Wide Web Conference, pp. 107–117. Elsevier Science, Amsterdam (1998)
3. Haveliwala, T., Kamvar, S., Jeh, G.: An Analytical Comparison of Approaches to Personalizing PageRank. In: Technical Report. Stanford University, Stanford (2003)
4. Langville, A.N., Meyer, C.D.: Deeper Inside PageRank. Internet Math. **1**(3), 335–380 (2004)
5. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web. In: Technical Report. Stanford University, Stanford (1998)

Paging

1985; Sleator, Tarjan, Fiat, Karp, Luby, McGeoch, Sleator, Young
1991; Sleator, Tarjan, Fiat, Karp, Luby, McGeoch, Sleator, Young

Rob van Stee
Department of Computer Science,
University of Karlsruhe, Karlsruhe, Germany

Keywords and Synonyms

Caching

Problem Definition

Computers generally have a small amount of fast memory to keep important data readily available. This is known as the *cache*. The question which is considered in this chapter is which pages should be kept in the cache when a new page is requested.

Formally, a two-level store of memory is considered. The cache can contain k pages, and the slow memory can contain n pages, where typically n is much larger than k. The input is a sequence of requests to pages. Whenever a requested page is not in the cache, the algorithm incurs a fault. The goal is to minimize the total number of page faults.

It is easy to give an optimal algorithm if the whole request sequence is known: on each fault, evict that page from the cache which is next requested the furthest in the future [2]. However, in practice, paging decisions need to be made without knowledge of the future. Thus an *online* algorithm is needed, which makes its decisions for each request based only on that request and previous requests.

Key Results

A major contribution of the paper of Sleator and Tarjan [6] was the idea of *competitive analysis*. In this type of analysis, the performance of an online algorithm is compared to that of an optimal offline algorithm OPT. Thus the offline algorithm knows the entire input and moreover it can use unbounded computational resources to find the best possible solution for this input.

Denote the cost of an algorithm ALG on an input sequence σ by $\text{ALG}(\sigma)$. An online algorithm $\mathcal{A}$ is called c-competitive if there exists a constant b such that on every request sequence σ,

$$\mathcal{A}(\sigma) \leq c \cdot \text{OPT}(\sigma) + b\,. \tag{1}$$

The competitive ratio of $\mathcal{A}$ is the smallest value of c such that $\mathcal{A}$ is c-competitive. This definition is very similar to that of the approximation ratio of approximation algorithms. However, it should be noted that there are no computational restrictions on the online algorithm. In particular, it is allowed to use exponential time to make its decisions. Thus the competitive ratio purely measures the performance loss that results from not knowing the future.

Using this definition, Sleator and Tarjan give tight bounds on the best competitive ratio which can be achieved by a deterministic algorithm. They show that two well-known algorithms both have a competitive ratio of k:

- FIFO (First In First Out), which on a fault evicts the page that was loaded into the cache the earliest
- LRU (Least Recently Used), which on a fault evicts the page that was *requested* least recently.

Additionally, they show that any deterministic algorithm has a competitive ratio of at least k, implying that k is the best value that can be achieved.

Fiat et al. [3] considered *randomized* paging algorithms. A randomized online algorithm is allowed to use random bits in its decision making. To measure its performance, consider the expectation of the cost for a particular input sequence and compare that to the optimal cost for that sequence. Thus a randomized online algorithm is c-competitive if there exists a constant b such that on every request sequence σ,

$$\mathbb{E}(\mathcal{A}(\sigma)) \leq c \cdot \text{OPT}(\sigma) + b\,. \tag{2}$$

Fiat et al. presented the marking algorithm. This algorithm marks pages that are requested. On a fault, an unmarked page is selected uniformly at random and evicted from the cache. When all pages are marked and a fault occurs, it unmarks all pages and then evicts one uniformly at random.

Fiat et al. showed that this algorithm is $2H_k$-competitive. Here H_k is the kth harmonic number: $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k}$. It is known that $\ln(k+1) \leq H_k \leq \ln(k) + 1$. They also showed that no randomized paging algorithm can have a competitive ratio less than H_k. Thus the marking algorithm is at most twice as bad as the best possible online algorithm (with regard to the competitive ratio). A randomized algorithm with competitive ratio exactly H_k was given by McGeoch and Sleator [4]. This algorithm is much more complicated than the marking algorithm.

Applications

Memory management has long been and continues to be a fundamentally important problem in computing systems. In particular, the question of how to manage a two-level or multilevel store of memory remains crucial to the performance of computers, from the simplest personal or game computer to the largest servers.

The study of the paging problem also was very important for the development of the whole area of online algorithms. The paper by Sleator and Tarjan formally introduced the concept of the competitive ratio as a performance measure for online algorithms. This ratio is in wide use today.

Open Problems

The problem as presented in this chapter is closed, since an upper and a lower bound of k for deterministic algorithms and an upper and a lower bound of H_k for randomized algorithms are obtained.

Variations of this problem continue to inspire new research. The basic problem has also been further studied, because the upper bound of k for LRU is disappointingly high and it is known from practice that LRU is "really" constant competitive. Recently, Panagiotou and Souza [5] managed to give a theoretical justification for this observation by formally restricting the input sequences to be closer to the ones that occur in practice. Additional justification for using LRU was given by Angelopoulos et al. [1], using a direct comparison of LRU to all other online algorithms.

Cross References

- ► Analyzing Cache Misses
- ► Deterministic Searching on the Line
- ► Online Paging and Caching

Recommended Reading

1. Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: On the separation and equivalence of paging strategies. In: Proceedings of the 18th Annual ACM–SIAM Symposium on Discrete Algorithms. ACM/SIAM, New York, Philadelphia (2007)
2. Belady, L.A.: A study of replacement algorithms for virtual storage computers. IBM Syst. J. **5**, 78–101 (1966)
3. Fiat, A., Karp, R., Luby, M., McGeoch, L.A., Sleator, D., Young, N.E.: Competitive paging algorithms. J. Algorithms **12**, 685–699 (1991)
4. McGeoch, L., Sleator, D.: A strongly competitive randomized paging algorithm. Algorithmica **6**(6), 816–825 (1991)
5. Panagiotou, K., Souza, A.: On adequate performance measures for paging. In: STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, pp. 487–496. ACM Press, New York, NY, USA (2006)
6. Sleator, D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**, 202–208 (1985)

Parallel Algorithms for Two Processors Precedence Constraint Scheduling

2003; Jung, Serna, Spirakis

Maria Serna
Department of Language & System Information, Technical University of Catalonia, Barcelona, Spain

Keywords and Synonyms

Optimal scheduling for two processors

Problem Definition

In the general form of *multiprocessor precedence scheduling problems* a set of n tasks to be executed on m processors is given. Each task requires exactly one unit of execution time and can run on any processor. A directed acyclic graph specifies the precedence constraints where an edge from task x to task y means task x must be completed before task y begins. A solution to the problem is a schedule of shortest length indicating when each task is started. The work of Jung, Serna, and Spirakis provides a parallel algorithm (on a PRAM machine) that solves the above problem for the particular case that $m = 2$, that is where there are two parallel processors.

The *two processor precedence constraint scheduling problem* is defined by a directed acyclic graph (dag) $G = (V, E)$. The vertices of the graph represent unit time tasks, and the edges specify precedence constraints among the tasks. If there is an edge from node x to node y then x is an *immediate predecessor* of y. *Predecessor* is the transitive closure of the relation immediate predecessor, and *successor* is its symmetric counterpart. A *two processor schedule* is an assignment of the tasks to time units $1, \ldots, t$ so that each task is assigned exactly one time unit, at most two tasks are assigned to the same time unit, and if x is a predecessor of y then x is assigned to a lower time unit than y. The length of the schedule is t. A schedule having minimum length is an *optimal* schedule. Thus the problem is the following:

Name	Two processor precedence constraint scheduling
Input	A directed acyclic graph
Output	A minimum length schedule preserving the precedence constraints.

Preliminaries

The algorithm assume that tasks are partitioned into levels as follows:

- **(i)** Every task will be assigned to only one level
- **(ii)** Tasks having no successors will be assigned to level 1 and
- **(iii)** For each level i, all tasks which are immediate predecessors of tasks in level i will be assigned to level $i + 1$.

Clearly topological sort will accomplish the above partition, and this can be done by an NC algorithm that uses $O(n^3)$ processors and $O(\log n)$ time, see [3]. Thus, from now on, it is assumed that a level partition is given as part of the input. For sake of convenience two special tasks, t_0 and t^* are added, in such a way that the original graph could be taught as the graph formed by all tasks that are

successors of t_0 and predecessors of t^*. Thus t_0 is a predecessor of all tasks in the system (actually an immediate predecessor of tasks in level the highest level $L(G)$) and t^* is a successor of all tasks in the system (an immediate successor of level 1 tasks).

Notice that if two tasks are at the same level they can be paired. But when x and y are at different levels, they can be paired only when neither of them is a predecessor of the other. Let $L(G)$ denote the number of levels in a given precedence graph G. A *level schedule* schedules tasks level by level. More precisely, suppose levels $L(G), \ldots, i+1$ have already been scheduled and there are k unscheduled tasks remaining on level i. When k is even, those tasks with are paired with each other. When k is odd, $k-1$ of the tasks are paired with each other, while the remaining task may (but not necessarily) be paired with a task from a lower level.

Given a level schedule level i *jumps to level* i' ($i' < i$) if the last time step containing a task from level i also contains a task from level i'. If the last task from level i is scheduled with an empty slot, it is said that level i *jumps to level* 0. The *jump sequence* of a level schedule is the list of levels jumped to. A *lexicographically first jump schedule* is a level schedule whose jump sequence is lexicographically greater than any other jump sequence resulting from a level schedule.

Given a graph G a *level partition* of G is a partition of the nodes in G into two sets in such a way that levels $0, \ldots, k$ are contained in one set (the upper part) denoted by U, and levels $k+1, \ldots, L$ in the other (the lower part) denoted by L. Given a graph G and a level i, the *i-partition* of G (or the partition at level i) is formed by the graphs U_i and L_i defined as U_i contains all nodes x such that level$(x) < i$ and L_i contains all nodes x with level$(x) > i$. Note that each i-partition determines two different level partitions depending on whether level i nodes are assigned to the upper or the lower part. A task $x \in U_i$ is called *free* with respect to a partition at level i if x has no predecessors in L_i.

Auxiliary Problems

The algorithm for the two processors precedence constraint scheduling problem uses as a building block an algorithm for solving a matching problem in a particular graph class.

A *full convex bipartite graph* G is a triple (V, W, E), where $V = \{v_1, \ldots, v_k\}$ and $W = \{w_1, \ldots, w_{k'}\}$ are disjoint sets of vertices. Furthermore the edge set E satisfies the following property: If $(v_i, w_j) \in E$ then $(v_q, w_j) \in E$ for all $q \geq i$. Thus, from now on it is assumed that the graph is connected.

A set $F \subseteq E$ is a *matching* in the graph $G = (V, W, E)$ iff no two edges in F have a common endpoint. A *maximal matching* is a matching that cannot be extended by the addition of any edge in G. A *lexicographically first maximal matching* is a maximal matching whose sorted list of edges is lexicographically first among all maximal matchings in G.

Key Results

When the number of processors m is arbitrary the problem is known to be NP-complete [8]. For any $m \geq 3$, the complexity is open [6]. Here the case of interest has been $m = 2$. For two processors a number of efficient algorithms has been given. For sequential algorithms see [2,4,5] among others. The first deterministic parallel algorithm was given by Helmbold and Mayr [7], thus establishing membership in the class NC. Previously [9] gave a randomized NC algorithm for the problem. Jung, Serna and Spirakis present a new parallel algorithm for the two processors scheduling problem that takes time $O(\log^2 n)$ and uses $O(n^3)$ processors on a CREW PRAM. The algorithm improves the number of processors of the algorithm given in [7] from $O(n^7 L(G)^2)$, where $L(G)$ is the number of levels in the precedence graph, to $O(n^3)$. Both algorithms compute a level schedule that has a lexicographically first jump sequence.

To match jumps with tasks it is used a solution to the problem of computing the lexicographically first matching for a special type of convex bipartite graphs, here called *full convex bipartite graphs*. A geometric interpretation of this problem leads to the discovery of an efficient parallel algorithm to solve it.

Theorem 1 *The lexicographically first maximal matching of full convex bipartite graphs can be computed in time* $O(\log n)$ *on a CREW PRAM with* $O(n^3/\log n)$ *processors, where n is the number of nodes.*

The previous algorithm is used to solve efficiently in parallel two related problems.

Theorem 2 *Given a precedence graph G, there is a PRAM parallel algorithm that computes all levels that jump to level 0 in the graph* L_i *and all tasks in level* $i-1$ *that can be scheduled together with a task in level i, for* $i = 1, \ldots, L(G)$, *using* $O(n^3)$ *processors and* $O(\log^2 n)$ *time.*

Theorem 3 *Given a level partition of a graph G together with the levels in the lower part in which one task remains to be matched with some other task in the upper part of the graph. There is a PRAM parallel algorithm that computes the corresponding tasks in time* $O(\log n)$ *using* $n^3/\log n$ *processors.*

With those building blocks the algorithm for two processor precedence constraint scheduling starts by doing some preprocessing and after that an adequate decomposition that insure that at each recursive call a number of problems of half size are solved in parallel. This recursive schema is the following:

Algorithm Schedule

0. Preprocessing
1. Find a level i such that $|U_i| \leq n/2$ and $|L_i| \leq n/2$.
2. Match levels that jump to free tasks in level i.
3. Match levels that jump to free tasks in U_i.
4. If level i (or $i+1$) remain unmatched try to match it with a non free task.
5. Delete all tasks used to match jumps.
6. Apply (1)–(5) in parallel to L_i and the modified U_i.

Algorithm **Schedule** stops whenever the corresponding graph has only one level.

The correction an complexity bounds for algorithm **Schedule** follows from the previous results, leading to:

Theorem 4 *There is an NC algorithm which finds an optimal two processors schedule for any precedence graph in time $O(\log^2 n)$ using $O(n^3)$ processors.*

Applications

A fundamental problem in many applications is to devise a proper schedule to satisfy a set of constrains. Assigning people to jobs, meetings to rooms, or courses to final exam periods are all different examples of scheduling problems. A key and critical algorithm in parallel processing is the one mapping tasks to processors. In the performance of such an algorithm relies many properties of the system, like load balancing, total execution time, etc. Scheduling problems differ widely in the nature of the constraints that must be satisfied, the type of processors, and the type of schedule desired.

The focus on precedence-constrained scheduling problems for directed acyclic graphs has a most direct practical application in problems arising in parallel processing. In particular in systems where computations are decomposed, prior to scheduling into approximately equal sized tasks and the corresponding partial ordering among them is computed. These constraints must define a directed acyclic graph, acyclic because a cycle in the precedence constraints represents a Catch situation that can never be resolved.

Open Problems

The parallel deterministic algorithm for the two processors scheduling problem presented here improves the number of processors of the Helmbold and Mayr algorithm for the problem [7]. However, the complexity bounds are far from optimal: recall that the sequential algorithm given in [5] uses time $O(e + n\alpha(n))$, where e is the number of edges in the precedence graph and $\alpha(n)$ is an inverse Ackermann's function. Such an optimal algorithm might have a quite different approach, in which the levelling algorithm is not used.

Interestingly enough computing the lexicographically first matching for full convex bipartite graphs is in NC, in contraposition with the results given in [1] which show that many problems defined through a lexicographically first procedure in the plane are P-complete. It is an interesting problem to show whether all these problems fall in NC when they are convex.

Cross References

- ▶ List Scheduling
- ▶ Maximum Matching
- ▶ Minimum Makespan on Unrelated Machines
- ▶ Shortest Elapsed Time First Scheduling
- ▶ Stochastic Scheduling
- ▶ Voltage Scheduling

Recommended Reading

1. Attallah, M., Callahan, P., Goodrich, M.: P-complete geometric problems. Int. J. Comput. Geom. Appl. **3**(4), 443–462 (1993)
2. Coffman, E.G., Graham, R.L.: Optimal scheduling for two processors systems. Acta Informatica **1**, 200–213 (1972)
3. Dekel, E., Nassimi, D., Sahni, S.: Parallel matrix and graph algorithms. SIAM J. Comput. **10**, 657–675 (1981)
4. Fujii, M., Kasami, T., Ninomiya, K.: Optimal sequencing of two equivalent processors. SIAM J. Comput. **17**, 784–789 (1969)
5. Gabow, H.N.: An almost linear time algorithm for two processors scheduling. J. ACM **29**(3), 766–780 (1982)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the theory of NP completeness. Freeman, San Francisco (1979)
7. Helmbold, D., Mayr, E.: Two processor scheduling is in NC. SIAM J. Comput. **16**(4), 747–756 (1987)
8. Ullman, J.D.: NP-complete scheduling problems. J. Comput. Syst. Sci. **10**, 384–393 (1975)
9. Vazirani, U., Vazirani, V.: Two-processor scheduling problem is in random NC. SIAM J. Comput. **18**(4), 1140–1148 (1989)

Parallel Connectivity and Minimum Spanning Trees

2001; Chong, Han, Lam

Tak-Wah Lam
Department of Computer Science,
University of Hong Kong, Hong Kong, China

Keywords and Synonyms

EREW PRAM algorithms for finding connected components and minimum spanning trees

Problem Definition

Given a weighted undirected graph G with n vertices and m edges, compute a minimum spanning tree (or spanning forest) of G on a parallel random access machine (PRAM) without concurrent write capability.

A minimum spanning tree of a graph is a spanning tree with the smallest possible sum of edge weights. Parallel random access machine (PRAM) is an abstract model for designing parallel algorithms and understanding the power of parallelism. In this model, processors (each being a random access machine) work in a synchronous manner and communicate through a shared memory. PARM can be further classified according to whether it is allowed for more than one processor to read and write into the same shared memory location simultaneously. The strongest model is CRCW (concurrent-read, concurrent-write) PRAM, and the weakest is EREW (exclusive-read, exclusive-write) PRAM. For an introduction to PRAM algorithms, one can refer to Karp and Ramachandran [8] and JáJá [5].

The input graph G is assumed to be given in the form of adjacency lists. Furthermore, isolated (degree-0) vertices are removed and hence it is assumed that $m \geq n$.

Key Results

The MST problem is related to the connected component (CC) problem, which is to find the connected components of an undirected graph. Sequential algorithms for solving the CC problem and the MST problem in $O(m)$ time and $O(m \log n)$ time, respectively, were known a few decades ago. A number of more efficient MST algorithms have since been published, the most recent of which is Pettie and Ramachandran's algorithm [9], which is provably optimal.

In the parallel context, both problems are often solved in a similar way. With respect to CRCW PRAM, the two problems can be solved using $O(\log n)$ time and $n + m$ processors (see, e. g., Cole and Vishkin [3]). Using randomization, $(n + m)/\log n$ processors are sufficient to solve these problems in $O(\log n)$ expected time [2,10].

For EREW PRAM, $O(\log^2 n)$ time algorithms for the CC and MST problems were developed in the early 80's. For a while, it was believed that the exclusive write models (including both concurrent read and exclusive read) could not overcome the $O(\log^2 n)$ time bound [8]. The first breakthrough was due to Johnson and Metaxas [6]; they devised $O(\log^{1.5} n)$ time algorithms for the CC problem and the MST problem. These results were soon improved to $O(\log n \log\log n)$ time by Chong and Lam. If randomization is allowed, the time complexity can be improved to $O(\log n)$ expected time and optimal work [7,10,11]. Finally, Chong, Han, and Lam [1] obtained an algorithm for MST (and CC) using $O(\log n)$ time and $n + m$ processors. This algorithm does not need randomization. Notice that $\Theta(\log n)$ is optimal since these graphs problems are at least as hard as computing the OR of n bits, and Cook et al. [4] have proven that the latter requires $\Omega(\log n)$ time on exclusive-write PRAM no matter how many processors are used.

Below is a sketch of some ideas for computing a minimum spanning tree in parallel without using concurrent write.

Without loss of generality, assume that the edge weights are all distinct. Thus, G has a unique minimum spanning tree, which is denoted by T^*_G. Let B be a subset of edges in G which contains no cycle. B induces a set of trees $F = \{T_1, T_2, \cdots, T_l\}$ in a natural sense—two vertices in G are in the same tree if they are connected by an edge of B. B is said to be a λ-forest if each tree $T \in F$ has at least λ vertices. For example, if B is the empty set then B is a 1-forest; a spanning tree is an n-forest.

Suppose that B is a λ-forest and its edges are all found in T^*_G. Then B can be augmented to give a 2λ-forest using a greedy approach: Let F' be an arbitrary subset of F including all trees $T \in F$ with fewer than 2λ vertices. For every tree in F', pick its minimum external edge (i. e., the smallest-weight edge connecting to a vertex outside the tree). Denote B' as this set of edges. It can be proven that B' consists of edges in T^*_G only, and $B \cup B'$ is a 2λ-forest. The above idea allows us to find T^*_G in $\lfloor \log n \rfloor$ stages as follows.

1. $B \leftarrow \phi$
2. **For** $i = 1$ to $\lfloor \log n \rfloor$ **do** /* Stage i */
   (a) Let F be the set of trees induced by B on G. Let F' be an arbitrary subset of F such that F' includes all trees $T \in F$ with fewer than 2^i vertices.
   (b) $B_i \leftarrow \{e \mid e$ is the minimum external edge of $T \in F'\}$; $B \leftarrow B \cup B_i$
3. **return** B

Different strategies for choosing the set F' in Step 1(a) may lead to different B_i's. Nevertheless, $B[1, i]$ is always a subset of T^*_G and induces a 2^i-forest. In particular, $B[1, \lfloor \log n \rfloor]$ induces exactly one tree, which is exactly T^*_G. Using standard parallel algorithmic techniques, each stage can be implemented in $O(\log n)$ time on EREW PRAM using a linear number of processors (see e. g. [5],). Therefore,

T_G^* can be found in $O(\log^2 n)$ time. In fact, most parallel algorithms for finding MST are based on a similar approach. These parallel algorithms are "sequential" in the sense that the computation of B_i starts only after B_{i-1} is available.

The $O(\log n)$-time EREW algorithm in [1], is based on some structural properties related to MST and can compute the B_i's in a more parallel fashion. In this algorithm, there are $\lfloor \log n \rfloor$ concurrent threads (a thread is simply a group of processors). For $1 \le i \le \lfloor \log n \rfloor$, Thread i aims at computing B_i, and it actually starts long before Thread $i-1$ has computed B_{i-1} and it receives the output of Threads 1 to $i-1$ (i. e., $B_1, \cdots, B_{i-1}$) incrementally. More specifically, the algorithm runs in $\lfloor \log n \rfloor$ supersteps, where each superstep lasts $O(1)$ time. Thread i delivers B_i at the end of the ith superstep. The computation of Thread i is divided into $\lfloor \log i \rfloor$ phases. Let us first consider a simple case when i is a power of two. Phase 1 of Thread i starts at the $(i/2+1)$th superstep, i. e., when $B_1, \cdots, B_{i/2}$ are available. Phase 1 takes no more than $i/4$ supersteps, ending at the $(i/2 + i/4)$th superstep. Phase 2 starts at the $(i/2 + i/4 + 1)$th superstep (i. e., when $B_{i/2+1}, \cdots, B_{i/2+i/4}$ are available) and uses $i/8$ supersteps. Each subsequent phase uses half as many supersteps as the preceding phase. The last phase (Phase $\log i$) starts and ends within the ith superstep; note that B_{i-} is available after $(i-1)$th superstep.

Applications

Finding connected components or MST is a key step in several parallel algorithms for other graph problems. For example, the Chong-Han-Lam algorithm implies an $O(\log n)$-time algorithm for finding ear decomposition and biconnectivity without using concurrent write.

Cross References

- ► Graph Connectivity
- ► Randomized Parallel Approximations to Max Flow

Recommended Reading

1. Chong, K.W., Han, Y., Lam, T.W.: Concurrent Threads and Optical Parallel Minimum Spanning Trees Algorithm. J. ACM **48**(2), 297–323 (2001)
2. Cole, R., Klein, P.N., Tarjan, R.E.: Finding minimum spanning forests in logarithmic time and linear work using random sampling. In: Proceedings of the 8th Annual ACM Symposium on Parallel Architectures and Algorithms, 1996, pp. 243–250
3. Cole, R., Vishkin, U.: Approximate and Exact Parallel Scheduling with Applications to List, Tree, and Graph Problems. In: Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 478–491
4. Cook, S.A., Dwork, C., Reischuk, R.: Upper and lower time bounds for parallel random access machines without simultaneous writes. SIAM J. Comput. **15**(1), 87–97 (1986)
5. JáJá, J.: An Introduction to Parallel Algorithms. Addison-Wesley (1992)
6. Johnson, D.B., Metaxas, P.: Connected Components in $O(\lg^{3/2} |V|)$ Parallel Time for the CREW PRAM. In: Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, 1991, pp. 688–697
7. Karger, D.R.: Random sampling in Graph Optimization Problems. Ph. D. thesis, Department of Computer Science, Stanford University (1995)
8. Karp, R.M., Ramachandran, V.: Parallel Algorithms for Shared-Memory Machines. In: Van Leeuwen Ed, J. (ed) Handbook of Theoretical Computer Science, vol. A, pp. (869–941). MIT Press, Massachusetts (1990)
9. Pettie, S. Ramachandran, V.: An Optimal Minimum Spanning Tree Algorithm. J. ACM **49**(1), 16–34 (2002)
10. Pettie, S., Ramachandran, V.: A randomized time-work optimal parallel algorithm for finding a minimum spanning forest. SIAM J. Comput. **31**(6), 1879–1895 (2002)
11. Poon, C.K., Ramachandran, V.: A randomized linear-work EREW PRAM algorithm to find a minimum spanning forest. Algorithmica **35**(3), 257–268 (2003)

Parameterized Algorithms for Drawing Graphs

2004; Dujmovic, Whitesides

Henning Fernau
Institute for Computer Science, University of Trier, Trier, Germany

Problem Definition

One-sided crossing minimization (OSCM) can be viewed as a specific form of drawing a bipartite graph $G = (V_1, V_2, E)$, where all vertices from partition V_i are assigned to the same line (also called layer) L_i in the plane, with L_1 and L_2 being parallel. The vertex assignment to L_1 is fixed, while that to L_2 is free and should be chosen in a way to minimize the number of crossings between etdes drawn as straight-line segments.

Notations

A graph G is described by its vertex set V and its edge set E, i. e., G=(V, E), with $E \subseteq V \times V$. The (open) *neighborhood* of a vertex v, denoted $N(v)$, collects all vertices that are adjacent to v. $N[v] = N(v) \cup \{v\}$ denotes the *closed neighborhood* of v. $\deg(v) = |N(v)|$ is the *degree* of v. For a vertex set S, $N(S) = \bigcup_{v \in S} N(v)$, and $N[S] = N(S) \cup S$. $G[S]$ denotes the graph induced by vertex set S, i. e., $G[S] = (S, E \cap (S \times S))$. A graph $G = (V, E)$ with vertex set

V and edge set $E \subseteq V \times V$ is *bipartite* if there is a partition of V into two sets V_1 and V_2 such that $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, and $E \subseteq V_1 \times V_2$. For clarity, $G = (V_1, V_2, E)$ is written in this case.

A *two-layer drawing* of a bipartite graph $G = (V_1, V_2, E)$ can be described by two linear orders $<_1$ on V_1 and $<_2$ on V_2. This drawing can be realized as follows: the vertices of V_1 are placed on a line L_1 (also called *layer*) in the order induced by $<_1$ and the vertices of V_2 are placed on a second layer L_2 (parallel to the first one) in the order induced by $<_2$; then, draw a straight-line segment for each edge $e = (u_1, u_2)$ in E connecting the points that represent u_1 and u_2, respectively. A *crossing* is a pair of edges $e = (u_1, u_2)$ and $f = (v_1, v_2)$ that cross in the realization of a two-layer drawing $(G, <_1, <_2)$. It is well-known that two edges cross if and only if $u_1 <_1 v_1$ and $v_2 <_2 u_2$; in other word, this notion is a purely combinatorical object, independent of the concrete realization of the two-layer drawing. $\mathrm{cr}(G, <_1, <_2)$ denotes the number of crossings in the described two-layer drawing. In the graph drawing context, it is of course desirable to draw graphs with few crossings. In its simplest (yet probably most important) form, the vertex order in one layer is fixed, and the aim is to minimize crossings by choosing an order of the second layer. Formally, this means:

Problem 1 (*k*–OSCM)
INPUT: *A simple n-vertex bipartite graph* $G = (V_1, V_2, E)$ *and a linear order* $<_1$ *on* V_1*, a nonnegative integer k (the parameter).*
OUTPUT: *If possible, a linear order* $<_2$ *on* V_2 *such that* $\mathrm{cr}(G, <_1, <_2) \leq k$*. If no such order exists, the algorithm should tell so.*

Given an instance $G = (V_1, V_2, E)$ and $<_1$ of OSCM and two vertices $u, v \in V_2$,

$$c_{uv} = \mathrm{cr}(G[N[\{u, v\}]], <_1 \cap (N(\{u, v\}) \times N(\{u, v\})), \{(u, v)\}) \; .$$

Hence, the closed neighborhoods of u and v are considered when assuming the ordering $u <_2 v$.

Consider the following as a running example:

Example 1 In Fig. 1, a concrete drawing of a bipartite graph is shown. Is this drawing optimal with respect to the number of crossings, assuming the ordering of the upper layer being fixed? At some points, more than two edges cross; in that case, a number is shown to count the crossings. All crossings are emphasized by a surrounding box.

Let us now compute the *crossing number matrix* (c_{uv}) for this graph.

c_{uv}	a	b	c	d	e
a	–	4	5	0	1
b	1	–	1	0	0
c	3	3	–	0	1
d	3	2	3	–	1
e	2	3	2	0	–

The number of crossings in the given drawing can be hence computed as

$$c_{ab} + c_{ac} + c_{ad} + c_{ae} + c_{bc} + c_{bd} + c_{be} + c_{cd} + c_{ce} + c_{de} = 13 \; .$$

Key Results

Exact exponential-time algorithms are mostly interesting when dealing with problems for which no polynomial-time algorithm is expected to exist.

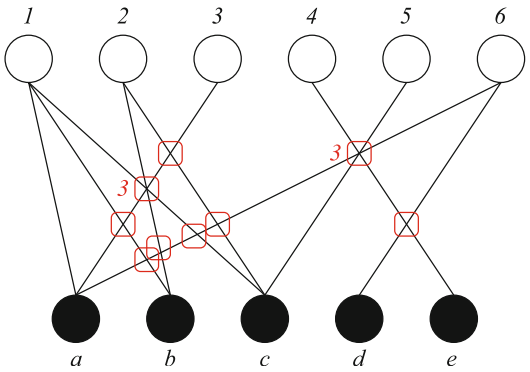
Theorem 1 ([6]) *The decision problem corresponding to k-OSCM is* $\mathcal{NP}$*-complete.*

In the following, to state the results, let $G = (V_1, V_2, E)$ be an instance of OSCM, where the ordering $<_1$ of V_1 is fixed.

It can be checked in polynomial time if an order of V_2 exists that avoids any crossings. This observation can be based on either of the following graph-theoretic characterizations:

Theorem 2 ([3]) $\mathrm{cr}(G, <_1, <_2) = 0$ *if and only if G is acyclic and, for every path* (x, a, y) *of G with* $x, y \in V_1$*, it holds: for all* $u \in V_1$ *with* $x <_1 u <_1 y$*, the only edge incident to u (if any) is (u, a).*

The previously introduced notion is crucial due to the following facts:



Parameterized Algorithms for Drawing Graphs, Figure 1
The running example for OSCM

Lemma 3 $\sum_{u,v\in V_2, u<_2 v} c_{uv} = \mathrm{cr}(G, <_1, <_2)$.

Theorem 4 ([9]) *If k is the minimum number of edge crossings in an OSCM instance $(G = (V_1, V_2, E), <_1)$, then*

$$\sum_{u,v\in V_2, u\neq v} \min\{c_{uv}, c_{vu}\} \leq k < 1.4664 \sum_{u,v\in V_2, u\neq v} \min\{c_{uv}, c_{vu}\} .$$

In fact, Nagamochi also presented an approximation algorithm with a factor smaller than 1.4664.

Furthermore, for any $u \in V_2$ with $\deg(u) > 0$, let l_u be the leftmost neighbor of u on L_1, and r_u be the rightmost neighbor of u. Two vertices $u, v \in V_2$ are called *unsuited* if there exists some $x \in N(u)$ with $l_v <_1 x <_1 r_v$, or there exists some $x \in N(v)$ with $l_u <_1 x <_1 r_u$. Otherwise, they are called *suited*. Observe that, for $\{u, v\}$ suited, $c_{uv} \cdot c_{vu} = 0$. Dujmović and Whitesides have shown:

Lemma 5 ([5]) *In any optimal ordering $<_2$ of the vertices of V_2, $u <_2 v$ is found if $r_u \leq_1 l_v$.*

This means that all suited pairs appear in their *natural ordering*.

This already allows us to formulate a first parameterized algorithm for OSCM, which is a simple search tree algorithm. In the course of this algorithm, a suitable ordering $<_2$ on V_2 is gradually constructed; when settling the ordering between u and v on V_2, $u <_2 v$ or $v <_2 u$ is *committed*. A *generalized instance* of OSCM therefore contains, besides the bipartite graph $G = (V_1, V_2, E)$, a partial ordering $<_2$ on V_2. A vertex $v \in V_2$ is *fully committed* if, for all $u \in V_2 \setminus \{u, v\}$, $\{u, v\}$ is committed.

Lemma 5 allows us to state the following rule:

RR1: For every pair of vertices $\{u, v\}$ from V_2 with $c_{uv} = 0$, commit $u <_2 v$. In the example, d would be fully committed by applying RR1, since the d-column in the crossing number matrix is all zeros; hence, ignore d in what follows.

Algorithm 1 is a simple search tree algorithm for OSCM that repeatedly uses Rule RR1.

Lemma 6 *OSCM can be solved in time $\mathcal{O}^*(2^k)$.*

Proof Before any branching can take place, the graph instance is reduced, so that every pair of vertices $\{u, v\}$ from V_2 which is not committed satisfies $\min\{c_{uv}, c_{vu}\} \geq 1$. Therefore, each recursive branch reduces the parameter by at least one. □

It is possible to improve on this very simple search tree algorithm. A first observation is that it is not necessary to branch at $\{x, y\} \subset V_2$ with $c_{xy} = c_{yx}$. This means two modifications to Algorithm 1:

- Line 5 should exclude $c_{xy} = c_{yx}$.
- Line 12 should arbitrary commit some $\{x, y\} \subset V_2$ that are not yet committed and recurse; only if all $\{x, y\} \subset V_2$ are committed, YES is to be returned.

These modifications immediately yield an $\mathcal{O}^*(1.6182^k)$ algorithm for OSCM. This is also the core of the algorithm proposed by Dujmović and Whitesides [5]. There, more details are discussed, as, for example:

- How to efficiently calculate all the crossing numbers c_{xy} in a preprocessing phase.
- How to integrate branch and cut elements in the algorithm that are surely helpful from a practical perspective.
- How to generalize the algorithm for instances that allow integer weights on the edges (multiple edges).

Further improvements are possible if one gives a deeper analysis of local patterns $\{x, y\} \in V_2$ such that $c_{xy}c_{yx} \leq 2$. This way, it has been shown:

Theorem 7 ([4]) *OSCM can be solved in time $\mathcal{O}^*(1.4656^k)$.*

A possible run of the improved search tree algorithm is displayed in Fig. 2, with the (optimal) outcome shown in Fig. 3.

Variants and Related Problems have been discussed in the literature.

1. Change the goal of the optimization: minimize the number of edges involved in crossings (One-layer planarization (OLP)). As observed in [7,10], Theorem 2 almost immediately leads to an $\mathcal{O}^*(3^k)$ algorithm for OLP that was subsequently improved down to $\mathcal{O}^*(2^k)$ in [10].
2. One could allow more degrees of freedom by considering two (or more) layer assignments at the same time. For both the crossing minimization and the planarization variants, parameterized algorithms are reported in [3,7,10].
3. One can consider other additional constraints on the drawings or the admissible orderings; in [8], parameterized algorithms for two-layer assignment problems are discussed where the admissible orderings are restricted by binary trees.

Applications

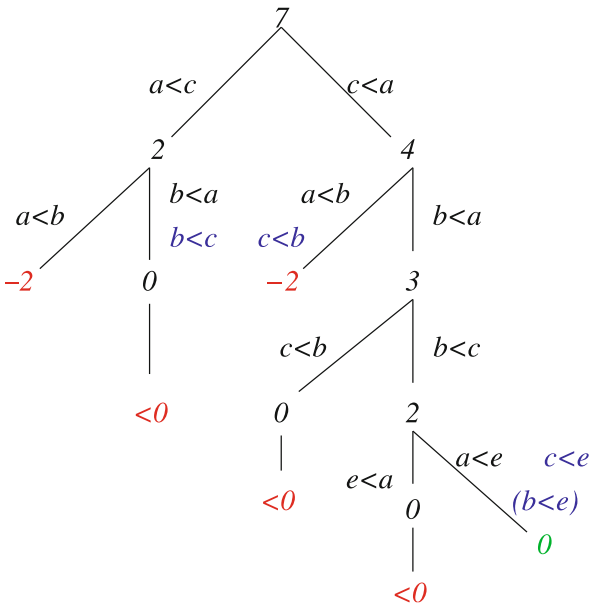
Besides seeing the question of drawing bipartite graphs as an interesting problem in itself, e. g., for nice drawings of relational diagrams, this question quite naturally shows up in the so-called Sugiyama approach to hierarchical graph drawing, see [12]. This very popular approach tack-

Require: a bipartite graph $G = (V_1, V_2, E)$, an integer k, a linear ordering $<_1$ on V_1, a partial ordering $<_2$ on V_2
Ensure: YES iff the given OSCM instance has a solution

repeat
  Exhaustively apply the reduction rules, adjusting $<_2$ and k accordingly.
  Determine the vertices whose order is settled by transitivity and adjust $<_2$ and k accordingly.
until there are no more changes to $<_2$ and to k
if $k < 0$ or $<_2$ contains both (x, y) and (y, x) **then**
  return NO.
else if $\exists\{x, y\} \subseteq V_2$: neither $x <_2 y$ nor $y <_2 x$ is settled **then**
  if OSCM-ST-simple$(G, k-1, <_1, <_2 \cup \{(x, y)\})$ **then**
    return YES
  else
    return OSCM-ST-simple$(G, k-1, <_1, <_2 \cup \{(y, x)\})$
  end if
else
  return YES
end if

Parameterized Algorithms for Drawing Graphs, Algorithm 1
A search tree algorithm solving OSCM, called OSCM-ST-simple

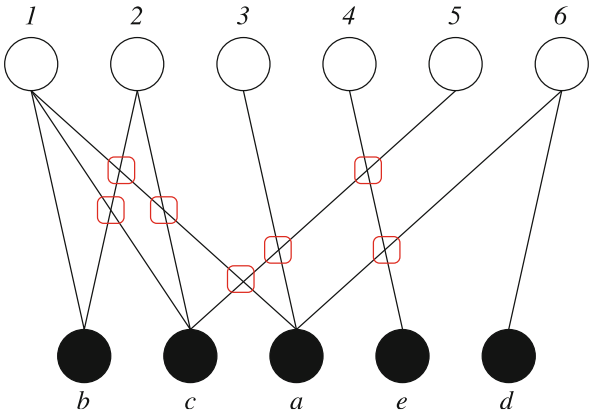
les the problem of laying out a hierarchical graph in three phases: (1) cycle removal (2) assignment of vertices to layers, (3) assignment of vertices to layers. The last phase is usually performed in a sweep-line fashion, intermediately solving many instances of OSCM. The third variant in the discussion above has important applications in computational biology.



Parameterized Algorithms for Drawing Graphs, Figure 2
A search tree example for OSCM

Open Problems

As with all exponential-time algorithms, it is always a challenge to further improve on the running times of the algorithms or to prove lower bounds on those running times under reasonable complexity theoretic assumptions. Let us notice that the tacit assumptions underlying the approach by parameterized algorithmics are well met in this application scenario: e. g., one would not accept drawings with



Parameterized Algorithms for Drawing Graphs, Figure 3
An optimal solution to the example instance

many crossings anyways (if such a situation is encountered in practice, one would switch to another way of representing the information); so, one can safely assume that the parameter is indeed small.

This is also true for other $\mathcal{NP}$-hard subproblems that relate to the Sugiyama approach. However, no easy solutions should be expected. For example, the DIRECTED FEEDBACK ARC SET PROBLEM [1] that is equivalent to the first phase is not known to admit a nice parameterized algorithm, see [2].

Experimental Results

Suderman [10] reports on experiments with nearly all problem variants discussed above, also see [11] for a better accessible presentation of some of the experimental results.

URL to Code

Suderman presents several JAVA applets related to the problems discussed in this article, see http://cgm.cs.mcgill.ca/~msuder/.

Cross References

Other parameterized search tree algorithms are explained in the contribution ► Vertex Cover Search Trees by Chen, Kanj, and Jia.

Recommended Reading

1. Chen, J., Liu, Y., Lu, S., O'Sullivan, B., Razgon, I.: A Fixed-Parameter Algorithm for the Directed Feedback Vertex Set Problem. In: 40th ACM Symposium on Theory of Computing STOC 2008, May 17–20, Victoria (BC), Canada (2008)
2. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Berlin (1999)
3. Dujmović, V., Fellows, M.R., Hallett, M., Kitching, M., Liotta, G., McCartin, C., Nishimura, N., Ragde, P., Rosamond, F.A., Suderman, M., Whitesides, S., Wood, D.R.: A fixed-parameter approach to 2-layer planarization. Algorithmica **45**, 159–182 (2006)
4. Dujmović, V., Fernau, H., Kaufmann, M.: Fixed parameter algorithms for one-sided crossing minimization revisited. In: Liotta G. (ed.) Graph Drawing, 11th International Symposium GD 2003. LNCS, vol. 2912, pp. 332–344. Springer, Berlin (2004). A journal version has been accepted to J. Discret. Algorithms, see doi: 10.1016/j.jda.2006.12.008
5. Dujmović, V., Whitesides, S.: An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. Algorithmica **40**, 15–32 (2004)
6. Eades, P., Wormald, N.C.: Edge crossings in drawings of bipartite graphs. Algorithmica **11**, 379–403 (1994)
7. Fernau, H.: Two-layer planarization: improving on parameterized algorithmics. J. Graph Algorithms Appl. **9**, 205–238 (2005)
8. Fernau, H., Kaufmann, M., Poths, M.: Comparing trees via crossing minimization. In: Ramanujam R., Sen S. (eds.) Foundations of Software Technology and Theoretical Computer Science FSTTCS 2005. LNCS, vol. 3821, pp. 457–469. Springer, Berlin (2005)
9. Nagamochi, H.: An improved bound on the one-sided minimum crossing number in two-layered drawings. Discret. Comput. Geom. **33**, 569–591 (2005)
10. Suderman, M.: Layered Graph Drawing. Ph. D. thesis, McGill University, Montréal (2005)
11. Suderman, M., Whitesides, S.: Experiments with the fixed-parameter approach for two-layer planarization. J. Graph Algorithms Appl. **9**(1), 149–163 (2005)
12. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Trans. Syst. Man Cybernet. **11**(2), 109–125 (1981)

Parameterized Matching

1993; Baker

MOSHE LEWENSTEIN
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Problem Definition

Parameterized strings, or *p-strings*, are strings that contain both ordinary symbols from an alphabet Σ and parameter symbols from an alphabet Π. Two equal-length p-strings s and s' are a parameterized match, or *p-match*, if one p-string can be transformed into the other by applying a one-to-one function that renames the parameter symbols. The following example of a p-match is one with both ordinary and parameter symbols. The ordinary symbols are in lowercase and the parameter symbols are in uppercase.

$$s = A\ b\ A\ b\ C\ A\ d\ b\ A\ C\ d\ d$$
$$s' = D\ b\ D\ b\ E\ D\ d\ b\ D\ E\ d\ d$$

In some of the problems to be considered it will be sufficient to solve for p-strings in which all symbols are parameter symbols, as this is the more difficult part of the problem. In other words, the case in which $\Sigma = \emptyset$. In this case the definition can be reformulated so that s and s' are a p-match if there exists a bijection $\pi : \Pi_s \to \Pi_{s'}$, such that $\pi(s) = s'$, where $\pi(s)$ is the renaming of each character of s via π.

The following problems will be considered. *Parameterized matching* – given a parameterized pattern p of length m and parameterized text t, find all locations i of a parameterized text t for which p p-matches $t_i \cdots t_{i+m-1}$, where $m = |p|$. The same problem is also considered in

two dimensions. *Approximate parameterized matching*– find all substrings of a parameterized text t that are approximate parameterized matches of a parameterized pattern p (to be fully defined later).

Key Results

Baker [4] introduced parameterized matching in the framework of her seminal work on discovering duplicate code within large programs for the sake of code minimization. An example of two code fragments that p-match taken from the X Windows system can be found in [4].

Parameterized Suffix Trees

In [4] and in the follow-up journal versions [6,7] a novel method was presented for parameterized matching by constructing *parameterized suffix trees*. The advantage of the parameterized suffix tree is that it supports indexing, i. e., one can preprocess a text and subsequently answer parameterized queries p in $O(|p|)$ time. In order to achieve parameterized suffix trees it is necessary to introduce the concept of a *predecessor string*. A *predecessor string* of a string s has at each location i the distance between i and the location containing the previous appearance of the symbol. The first appearance of each symbol is replaced with a 0. For example, the predecessor string of *aabbaba* is 0, 1, 0, 1, 3, 2, 2. A simple and well-known fact is that:

Observation [7] s and s' p-match if and only if they have the same predecessor string.

Notice that this implies transitivity of parameterized matching, since if s and s' p-match and s' and s'' p-match then, by the observation, s and s' have the same predecessor string and, likewise, s' and s'' have the same predecessor string. This implies that s and s'' have the same predecessor string and hence, by the observation, p-match.

Moreover, one may also observe that if r is a prefix of s then the predecessor string of r, by definition, is exactly the $|r|$-length prefix of the predecessor string of s. Hence, similar to regular pattern matching, a parameterized pattern p p-matches at location i of t if and only if the $|p|$-length predecessor string of p is equal to the $|p|$-length prefix of the predecessor string of the suffix $t_i \cdots t_n$. Combining these observations it is natural to do as follows; create a (parameterized suffix) tree with a leaf for each suffix where the path from the root to the leaf corresponding to a given suffix will have its predecessor string labeling the path. Branching in the parameterized suffix tree, as with suffix trees, occurs according to the labels of the predecessor strings. See [4,6,7] for an example.

Baker's method essentially mimics the McCreight suffix tree construction [18]. However, while the suffix tree and the parameterized suffix tree are very similar, there is a slight hitch. A strong component of the suffix tree construction is the suffix link. This is used for the construction and, sometimes, for later pattern searches. The suffix link is based on the *distinct right context* property, which does not hold for the parameterized suffix tree. In fact, the node that is pointed to by the suffix link may not even exist. The main parts of [6,7] are dedicated to circumventing this problem.

In [7] Baker added the notion of "bad" suffix links, which point to the vertex just above, i. e., closer to the root than the desired place, and of updating them with a lazy evaluation when they are used. The algorithm runs in time $O(n|\Pi|\log|\Sigma|)$. In [6] (which is chronologically later than [7] despite being the first to appear) Baker changed the definition of "bad" suffix links to point to just below the desired place. This turns out to have nice properties and one can use more sophisticated data structures to improve the construction time to $O(n(|\Pi| + \log|\Sigma|))$.

Kosaraju [16] made a careful analysis of Baker's properties utilized in the algorithm of [6] which suffer from the $|\Pi|$ factor. He pointed out two sources for this large factor. He handled these two issues by using a concatenable queue and maintaining it in a lazy manner. This is sufficient to reduce the $|\Pi|$ factor to a $\log|\Pi|$ factor, yielding an algorithm of time $O(n(\log|\Pi| + \log|\Sigma|))$.

Obviously if the alphabet or symbol set is large the construction time may be $O(n \log n)$. Cole and Hariharan [9] showed how to construct the parameterized suffix trees in randomized $O(n)$ time for alphabets and parameters taken from a polynomially sized range, e. g., $[1, \cdots, n^c]$. They did this by adding additional nodes to the tree in a back-propagation manner which is reminiscent of fractional cascading. They showed that this adds only $O(n)$ nodes and allows the updating of the missing suffix links. However, this causes other problems and one may find the details of how this is handled in their paper.

More Methods for Parameterized Matching

Obviously the parameterized suffix tree efficiently solves the parameterized matching problem. Nevertheless, a couple of other results on parameterized matching are worth mentioning.

First, in [6] it was shown how to construct the parameterized suffix tree for the pattern and then to run the parameterized text through it, giving an algorithm with $O(m)$ space instead of $O(n)$.

Amir et al. [2] presented a simple method to solve the parameterized matching problem by mimicking the algorithm of Knuth, Morris and Pratt. Their algorithm works in $O(n * \min(\log |\Pi|, m))$ time independent of the alphabet size ($|\Sigma|$). Moreover, they proved that the log factor cannot be avoided for large symbol sets.

In [5] parameterized matching was solved with a Boyer–Moore type algorithm. In [10] the problem was solved with a Shift–Or type algorithm. Both handle the average case efficiently. In [10] emphasis was also put on the case of multiple parameterized matching, which was previously solved in [14] with an Aho–Corasick automaton-style algorithm.

Two-Dimensional Parameterized Matching

Two-dimensional parameterized matching arises in applications of image searching; see [13] for more details. Two-dimensional parameterized matching is the natural extension of parameterized matching where one seeks p-matches of a two-dimensional parameterized pattern p within a two-dimensional parameterized text t. It must be pointed out that classical methods for two-dimensional pattern matching, such as the L-suffix tree method, fail for parameterized matching. This is because known methods tend to cut the text and pattern into pieces to avoid going out of boundaries of the pattern. This is fine because each pattern piece can be individually evaluated (checked for equality) to a text piece. However, in parameterized matching there is a strong dependency between the pieces.

In [1] an innovative solution for the problem was given based on a collection of linearizations of the pattern and text with the property to be currently described. Consider a linearization. Two elements with the same character, say 'a,' in the pattern are defined to be neighbors if there is no other 'a' between them in this linearization. Now take all the 'a's of the pattern and create a graph G_a with 'a's as the nodes and edges between two if they are neighbors in some linearization. We say that two 'a's are chained if there is a path from one to the other in G_a. Applying one-dimensional parameterized matching on these linearizations ensures that any two elements that are chained will be evaluated to map to the same text value (the parameterized property). A collection of linearizations has the *fully chained* property if every two locations in p with the same character are chained. It was shown in [1] that one can obtain a collection of $\log m$ linearizations that is fully chained and that does not exceed pattern boundary limits. Each such linearization is solved with a convolution-based pattern matching algorithm. This takes $O(n^2 \log m)$ time for each linearization, where the text size is n^2. Hence, overall the time is $O(n^2 \log^2 m)$.

A different solution was proposed in [13], where it was shown that it is possible to solve the problem in $O(n^2 + m^{2.5}\text{polylog } m)$, where the text size is $O(n^2)$ and the pattern size is $O(m^2)$. Clearly, this is more efficient for large texts.

Approximate Parameterized Matching

Our last topic relates to parameterized matching in the presence of errors. Errors occur in the various applications and it is natural to consider parameterized matching with the Hamming distance metric or the edit distance metric.

In [8] the parameterized matching problem was considered in conjunction with the edit distance. Here the definition of edit distance was slightly modified so that the edit operations are defined to be insertion, deletion and parameterized replacements, i. e., the replacement of a substring with a string that p-matches it. An algorithm for finding the "parameterized edit distance" of two strings was devised whose efficiency is close to the efficiency of the algorithms for computing the classical edit distance.

However, it turns out that the operation of parameterized replacement relaxes the problem to an easier problem. The reason that the problem becomes easier is that two substrings that participate in two parameterized replacements are independent of each other (in the parameterized sense).

A more rigid, but more realistic, definition for the Hamming distance variant was given in [3]. For a pair of equal-length strings s and s' and a bijection π defined on the alphabet of s, the π-mismatch is the Hamming distance between the image under π of s and s'. The minimal π-mismatch over all bijections π is the approximate parameterized match. The problem considered in [3] is to find for each location i of a text t the approximate parameterized match of a pattern p with the substring beginning at location i. In [3] the problem was defined and linear-time algorithms were given for the case where the pattern is binary or the text is binary. However, this solution does not carry over to larger alphabets.

Unfortunately, under this definition the methods for classical string matching with errors for the Hamming distance, also known as pattern matching with mismatches, seem to fail. Following is an outline of a classical method [17] for pattern matching with mismatches that uses suffix trees.

The pattern is compared separately with each suffix of the text, beginning at locations $1 \leq i \leq n - m + 1$. Using a suffix tree of the text and precomputed longest common

ancestor information (which can be computed once in linear time [11]), one can find the longest common prefix of the pattern and the corresponding suffix (in constant time). There must be a mismatch immediately afterwards. The algorithm jumps over the mismatch and repeats the process, taking into consideration the offsets of the pattern and suffix.

When attempting to apply this technique to a parameterized suffix tree, it fails. To illustrate this, consider the first matching substring (up until the first error) and the next matching substring (after the error). Both of these substrings p-match the substring of the text that they are aligned with. However, it is possible that combined they do not form a p-match. See the example below. In the example *abab* p-matches *cdcd* followed by a mismatch and subsequently followed by *abaa* p-matching *efee*. However, different π's are required for the local p-matches. This example also emphasizes why the definition of [8] is a simplification. Specifically, each local p-matching substring is one replacement, i. e., *abab* with *cdcd* is one replacement and *abaa* with *efee* is one more replacement. However, the definition of [3] captures the globality of the parameterized matching, not allowing, in this case, *abab* to p-match to two different substrings.

$$p = a\ b\ a\ b\ a\ a\ b\ a\ a \cdots$$
$$t = \cdots\ c\ d\ c\ d\ d\ e\ f\ e\ e \cdots$$

In [12] the problem of *parameterized matching with k mismatches* was considered. The parameterized matching problem with k mismatches seeks all locations i in text t where the minimal π-mismatch between p to $t_i \cdots t_{i+m-1}$ has at most k mismatches. An $O(nk^{1.5} + mk \log m)$ time algorithm was presented in [12]. At the base of the algorithm, i. e., for the case where $|p| = |t| = m$, an $O(m + k^{1.5})$ algorithm is used based on maximum matching algorithms. Then the algorithm uses a doubling scheme to handle the growing distance between potential parameterized matches (with at most k mismatches). Also shown in [12] is a strong relationship between maximum matching algorithms in sparse graphs and parameterized matching with k errors.

The rigid, but more realistic, definition for the Hamming distance version given in [3] can be naturally extended to the edit distance. Lately, it was shown that this problem is nondeterministic polynomial-time complete [15].

Applications

Parameterized matching has applications in code duplication detection in programming languages, in homework plagiarism detection, and in image processing, among others [1,4].

Cross References

- ▶ Multidimensional String Matching
- ▶ Sequential Approximate String Matching
- ▶ Sequential Exact String Matching
- ▶ Suffix Tree Construction in Hierarchical Memory
- ▶ Suffix Tree Construction in RAM

Recommended Reading

1. Amir, A., Aumann, Y., Cole, R., Lewenstein, M., Porat, E.: Function matching: Algorithms, applications and a lower bound. In: Proc. of the 30th International Colloquium on Automata, Languages and Programming (ICALP), 2003 pp. 929–942
2. Amir, A., Farach, M., Muthukrishnan, S.: Alphabet dependence in parameterized matching. Inf. Process. Lett. **49**, 111–115 (1994)
3. Apostolico, A., Erdös, P., Lewenstein, M.: Parameterized matching with mismatches. J. Discret. Algorithms **5**(1), 135–140 (2007)
4. Baker, B.S.: A theory of parameterized pattern matching: algorithms and applications. In: Proc. 25th Annual ACM Symposium on the Theory of Computation (STOC), 1993, pp. 71–80
5. Baker, B.S.: Parameterized pattern matching by Boyer-Moore-type algorithms. In: Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1995, pp. 541–550
6. Baker, B.S.: Parameterized pattern matching: Algorithms and applications. J. Comput. Syst. Sci. **52**(1), 28–42 (1996)
7. Baker, B.S.: Parameterized duplication in strings: Algorithms and an application to software maintenance. SIAM J. Comput. **26**(5), 1343–1362 (1997)
8. Baker, B.S.: Parameterized diff. In: Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1999, pp. 854–855
9. Cole, R., Hariharan, R.: Faster suffix tree construction with missing suffix links. In: Proc. 32nd ACM Symposium on Theory of Computing (STOC), 2000 pp. 407–415
10. Fredriksson, K., Mozgovoy, M.: Efficient parameterized string matching. Inf. Process. Lett. **100**(3), 91–96 (2006)
11. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestor. J. Comput. Syst. Sci. **13**, 338–355 (1984)
12. Hazay, C., Lewenstein, M., Sokol, D.: Approximate parameterized matching. ACM Trans. Algorithms **3**(3) (2007)
13. Hazay, C., Lewenstein, M., Tsur, D.: Two dimensional parameterized matching. In: Proc. of 16th Symposium on Combinatorial Pattern Matching (CPM), 2005, pp. 266–279
14. Idury, R.M., Schäffer, A.A.: Multiple matching of parametrized patterns. Theor. Comput. Sci. **154**(2), 203–224 (1996)
15. Keller, O., Kopelowitz, T., Lewenstein, M.: Parameterized LCS and edit distance are NP-Complete. Manuscript
16. Kosaraju, S.R.: Faster algorithms for the construction of parameterized suffix trees. In: Proc. 36th Annual Symposium on Foundations of Computer Science (FOCS), 1995, pp. 631–637
17. Landau, G.M., Vishkin, U.: Fast string matching with k differences. J. Comput. Syst. Sci. **37**(1), 63–78 (1988)
18. McCreight, E.M.: A space-economical suffix tree construction algorithm. J. ACM **23**, 262–272 (1976)

Parameterized SAT

2003; Szeider

Stefan Szeider
Department of Computer Science, Durham University, Durham, UK

Keywords and Synonyms

Structural parameters for SAT

Problem Definition

Much research has been devoted to finding classes of propositional formulas in conjunctive normal form (CNF) for which the recognition problem as well as the propositional satisfiability problem (SAT) can be decided in polynomial time. Some of these classes form infinite chains $C_1 \subset C_2 \subset \cdots$ such that every CNF formula is contained in some C_k for k sufficiently large. Such classes are typically of the form $C_k = \{F \in \text{CNF}: \pi(F) \leq k\}$, where π is a computable mapping that assigns to CNF formulas F a non-negative integer $\pi(F)$; we call such a mapping a *satisfiability parameter*. Since SAT is an NP-complete problem (actually, the first problem shown to be NP-complete [1]), we must expect that, the larger k, the longer the worst-case running times of the polynomial-time algorithms that recognize and decide satisfiability of formulas in C_k. Whence there is a certain tradeoff between the generality of classes and the performance guarantee for the corresponding algorithms. Szeider [12] initiates a broad investigation of this tradeoff in the conceptional framework of *parameterized complexity* [2,3,6]. This investigation draws attention to satisfiability parameters π for which the following holds: recognition and satisfiability decision of formulas F with $\pi(F) \leq k$ can be carried out in *uniform polynomial time*, that is, by algorithms with running time bounded by a polynomial whose order is independent of k (albeit, possibly involving a constant factor that is exponential in k). If a satisfiability parameter π allows satisfiability decision in uniform polynomial time, we say that *SAT is fixed-parameter tractable with respect to* π.

Satisfiability Parameters Based on Graph Invariants

One can define satisfiability parameters by means of certain graphs associated with CNF formulas. The *primal graph* of a CNF formula is the graph whose vertices are the variables of the formula; two variables are joined by an edge if the variables occur together in a clause. The *incidence graph* of a CNF formula is the bipartite graph whose vertices are the variables and clauses of the formula; a variable and a clause are joined by an edge if the variable occurs in the clause.

Satisfiability Parameters Based on Backdoor Sets

The concept of backdoor sets [13] gives rise to several interesting satisfiability parameters. Let C be a class of CNF formulas. A set B of variables of a CNF formula F is a *strong C-backdoor set* if for every partial truth assignment $\tau: B \to \{\text{true}, \text{false}\}$, the restriction of F to τ belongs to C. Here, the restriction of F to τ is the CNF formula obtained from F by removing all clauses that contain a literal that is true under τ and by removing from the remaining clauses all literals that are false under τ.

Key Results

Theorem 1 (Gottlob, Scarcello, and Sideri [4]) *SAT is fixed-parameter tractable with respect to the treewidth of primal graphs.*

Several satisfiability parameters that generalize the treewidth of primal graphs, such as the *treewidth and clique-width of incidence graphs*, have been studied [5,10,12].

The *maximum deficiency* of a CNF formula F is the number of clauses remaining exposed by a maximum matching of the incidence graph of F.

Theorem 2 (Szeider [11]) *SAT is fixed-parameter tractable with respect to maximum deficiency.*

A CNF formula is *minimal unsatisfiable* if it is unsatisfiable but removing any of its clauses makes it satisfiable. Recognition of minimal unsatisfiable formulas is DP-complete [9].

Corollary 1 (Szeider [11]) *Recognition of minimal unsatisfiable CNF formulas is fixed-parameter tractable with respect to the difference between the number of clauses and the number of variables.*

Theorem 3 (Nishimura, Ragde, and Szeider [7]) *SAT is fixed-parameter tractable with respect to the size of strong* Horn*-backdoor sets and with respect to the size of strong* 2CNF*-backdoor sets.*

Applications

Satisfiability provides a powerful and general formalism for solving various important problems including hardware and software verification and planning. Instances stemming from applications usually contain a "hidden

structure" (see, e. g. [13]). The satisfiability parameters considered above are designed to make this hidden structure explicit in the form of small values for the parameter. Thus, satisfiability parameters are a way to make the hidden structure accessible to an algorithm.

Open Problems

A new line of research is concerned with the identification of further satisfiability parameters that allow a fixed-parameter tractable SAT decision are more general than the known parameters and apply well to real-world problem instances.

Cross References

- ► Maximum Matching
- ► Treewidth of Graphs

Recommended Reading

1. Cook, S.A.: The complexity of theorem-proving procedures. In: Proc. 3rd Annual Symp. on Theory of Computing, Shaker Heights, OH 1971, pp. 151–158
2. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science. Springer, Berlin (1999)
3. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science, vol. XIV. An EATCS Series. Springer, Berlin (2006)
4. Gottlob, G., Scarcello, F., Sideri, M.: Fixed-parameter complexity in AI and nonmonotonic reasoning. Artif. Intell. **138**, 55–86 (2002)
5. Gottlob, G., Szeider, S.: Fixed-parameter algorithms for artificial intelligence, constraint satisfaction, and database problems. Comput. J., Special Issue on Parameterized Complexity, Advanced Access (2007)
6. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms, Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, Oxford, UK (2006)
7. Nishimura, N., Ragde, P., Szeider, S.: Detecting backdoor sets with respect to Horn and binary clauses. In: Informal proceedings of SAT 2004, 7th International Conference on Theory and Applications of Satisfiability Testing, Vancouver, BC, Canada, 10–13 May 2004, pp. 96–103
8. Nishimura, N., Ragde, P., Szeider, S.: Solving SAT using vertex covers. Acta Inf. **44**(7–8), 509–523 (2007)
9. Papadimitriou, C.H., Wolfe, D.: The complexity of facets resolved. J. Comput. Syst. Sci. **37**, 2–13 (1988)
10. Samer, M., Szeider, S.: Algorithms for propositional model counting. In: Proceedings of LPAR 2007, 14th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Yerevan, Armenia, 15–19 October 2007. Lecture Notes in Computer Science, vol. 4790, pp. 484–498. Springer, Berlin (2007)
11. Szeider, S.: Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable. J. Comput. Syst. Sci. **69**, 656–674 (2004)
12. Szeider, S.: On fixed-parameter tractable parameterizations of SAT. In: Giunchiglia, E., Tacchella, A. (eds.) Theory and Applications of Satisfiability, 6th International Conference, SAT 2003, Selected and Revised Papers. Lecture Notes in Computer Science, vol. 2919, pp. 188–202. Springer, Berlin (2004)
13. Williams, R., Gomes, C., Selman, B.: On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search, In: informal proceedings of SAT 2003 (Sixth International Conference on Theory and Applications of Satisfiability Testing, 5–8 May 2003, S. Margherita Ligure – Portofino, Italy), 2003, pp. 222–230

Pattern Matching

- ► Multidimensional Compressed Pattern Matching
- ► Two Dimensional Scaled Pattern Matching

Peer to Peer

- ► P2P

Peptide De Novo Sequencing with MS/MS

2005; Ma, Zhang, Liang

Bin Ma
Department of Computer Science, University of Western Ontario, London, ON, Canada

Keywords and Synonyms

De novo sequencing; Peptide sequencing

Problem Definition

De novo sequencing arises from the identification of peptides by using tandem mass spectrometry (MS/MS). A peptide is a sequence of amino acids in biochemistry and can be regarded as a string over a finite alphabet from a computer scientist's point of view. Each letter in the alphabet represents a different kind of amino acid, and is associate with a mass value. In the biochemical experiment, a tandem mass spectrometer is utilized to fragment many copies of the peptide into pieces, and to measure the mass values (in fact, the mass to charge ratios) of the fragments simultaneously. This gives a tandem mass spectrum. Since different peptides normally produce different spectra, it is possible, and now a common practice, to deduce the amino acid sequence of the peptide from its spectrum. Often this deduction involves the searching in a database for a peptide that can possibly produce the spectrum. But in many cases such a database does not exist or is not complete, and the calculation has to be done without looking

for a database. The latter approach is called *de novo* sequencing.

A general form of *de novo* sequencing problems is described in [2]. First, a score function $f(P, S)$ is defined to evaluate the pairing of a peptide P and a spectrum S. Then the *de novo* sequencing problem seeks for a peptide P such that $f(P, S)$ is maximized for a given spectrum S.

When the peptide is fragmented in the tandem mass spectrometer, many types of fragments can be generated. The most common fragments are the so called b-ions and y-ions. B-ions correspond to the prefixes of the peptide sequence, and y-ions the suffixes. Readers are referred to [8] for the biochemical details of the MS/MS experiments and the possible types of fragment ions. For clarity, in what follows only b-ions and y-ions are considered, and the *de novo* sequencing problem will be formulated as a pure computational problem.

A spectrum $S = \{(x_i, h_i)\}$ is a set of peaks, each has a mass value x_i and an intensity value h_i. A peptide $P = a_1 a_2 \dots a_n$ is a string over a finite alphabet Σ. Each $a \in \Sigma$ is associated with a positive mass value $m(a)$. For any string $t = t_1 t_2 \dots t_k$, denote $m(t) = \sum_{i=1}^{k} m(t_i)$. The mass of a length-$i$ prefix (b-ion) of P is defined as $b_i = c_b + m(a_1 a_2 \dots a_i)$. The mass of a length-$i$ suffix (y-ion) of P is defined as $y_i = c_y + m(a_{k-i+1} \dots a_{k-1} a_k)$. Here c_b and c_y are two constants related to the nature of the MS/MS experiments. If the mass unit used for measuring each amino acid is dalton, then $c_b = 1$ and $c_y = 19$.

Let δ be a mass error tolerance that is associated with the mass spectrometer. For mass value m, the peaks *matched* by m is defined as $D(m) = \{(x_i, h_i) \in S \,|\, |x_i - m| \le \delta\}$. The general idea of *de novo* sequencing is to maximize the number and intensities of the peaks matched by all b and y ions. Normally, δ is far less than the minimum mass of an amino acid. Therefore, for different i and j, $D(b_i) \cap D(b_j) = \emptyset$ and $D(y_i) \cap D(y_j) = \emptyset$. However, $D(b_i)$ and $D(y_j)$ may share common peaks. So, if not defined carefully, a peak may be counted twice in the score function. There are two different definitions of *de novo* sequencing problem, corresponding to two different ways of handling this situation.

Definition 1 (Anti-symmetric de novo sequencing)
Instance: A spectrum S, a mass value M, and an error tolerance δ.
Solution: A peptide P such that $m(P) = M$, and $D(b_i) \cap D(y_j) = \emptyset$ for any i, j.
Objective: Maximize $\sum_{k=1}^{n} \sum_{(x_i, h_i) \in D(b_k) \cup D(y_k)} h_i$.

This definition discards the peptides that gives a pair of b_i and y_j with similar mass values, because this happens rather infrequently in practice. Another definition allows the peptides to have pairs of b_i and y_j with similar mass values. However, when a peak is matched by multiple ions, it is counted only once. More precisely, define the matched peaks by P as

$$D(P) = \bigcup_{i=1}^{n} (D(b_i) \cup D(y_i)) .$$

Definition 2 (De novo sequencing)
Instance: A spectrum S, a mass value M, and an error tolerance δ.
Solution: A peptide P such that $m(P) = M$.
Objective: Maximize $f(P, S) = \sum_{(x_i, h_i) \in D(P)} h_i$.

Key Results

Anti-symmetric de novo sequencing was studied in [1,2]. These studies convert the spectrum into a spectrum graph. Each peak in the spectrum generates a few of nodes in the spectrum graph, corresponding to the different types of ions that may produce the peak. Each edge in the graph indicates that the mass difference of the two adjacent nodes is approximately the mass of an amino acid, and the edge is labeled with the amino acid. When at least one of each pair of b_i and y_{n-i} matches a peak in the spectrum, the *de novo* sequencing problem is reduced to the finding of the "anti-symmetric" longest path in the graph. A dynamic programming algorithm for such purpose was published in [1].

Theorem 1 ([1]) *The longest anti-symmetric path in a spectrum graph $G = \langle V, E \rangle$ can be found in $O(|V||E|)$ time.*

Under definition 2, *de novo* sequencing was studied in [5] and a polynomial time algorithm was provided. The algorithm is again a dynamic programming algorithm. For two mass values (m, m'), the dynamic programming calculates an optimal pair of prefix Aa and suffix $a'A'$, such that

1. $m(Aa) = m$ and $m(a'A') = m'$.
2. Either $c_b + m(A) < c_y + m(a'A') \le c_b + m(Aa)$ or $c_y + m(A') \le c_b + m(A) < c_y + m(a'A')$.

The calculation for (m, m') is based on the optimal solutions of smaller mass values. Because of the second above requirement, it is proved in [5] that not all pairs of (m, m') are needed. This is used to speed up the algorithm. A carefully designed strategy can eventually output a prefix and a suffix so that their concatenation form the optimal solution of the *de novo* sequencing problem. More specifically, the following theorem holds.

Theorem 2 ([6]) *The de novo sequencing problem has an algorithm that gives the optimal peptide in $O(|\Sigma| \times \delta \times \max_{a \in \Sigma} m(a) \times M)$.*

Because $|\Sigma|$, δ, $\max_{a\in\Sigma} m(a)$ are all constants, the algorithm in fact runs in linear time with a large coefficient.

Although the above algorithms are designed to maximize the total intensities of the matched peaks, they can be adapted to work on more sophisticated score functions. Some studies of other score functions can be found in [2,3,4,6]. Some of these score functions require new algorithms.

Applications

The algorithms have been implemented into software programs to assist the analyses of tandem mass spectrometry data. Software using the spectrum graph approach includes Sherenga [2]. The *de novo* sequencing algorithm under the second definition was implemented in PEAKS [6]. More complete lists of the *de novo* sequencing software and their comparisons can be found in [7,9].

URL to Code

PEAKS free trial version is available at http://www.bioinfor.com/.

Recommended Reading

1. Chen, T., Kao, M.-Y., Tepel, M., Rush J., Church, G.: A dynamic programming approach to de novo peptide sequencing via tandem mass spectrometry. J. Comput. Biol. **8**(3), 325–337 (2001)
2. Dančík, V., Addona, T., Clauser, K., Vath, J., Pevzner, P.: De novo protein sequencing via tandem mass-spectrometry. J. Comput. Biol. **6**, 327–341 (1999)
3. Fischer, B., Roth, V., Roos, F., Grossmann, J., Baginsky, S., Widmayer, P., Gruissem, W., Buhmann J.: NovoHMM: A Hidden Markov Model for de novo peptide sequencing. Anal. Chem. **77**, 7265–7273 (2005)
4. Frank, A., Pevzner, P.: Pepnovo: De novo peptide sequencing via probabilistic network modeling. Anal. Chem. **77**, 964–973 (2005)
5. Ma, B., Zhang, K., Liang, C.: An effective algorithm for the peptide de novo sequencing from MS/MS spectrum. J. Comput. Syst. Sci. **70**, 418–430 (2005)
6. Ma, B., Zhang, K., Lajoie, G., Doherty-Kirby, A., Hendrie, C., Liang, C., Li, M.: PEAKS: Powerful software for peptide de novo sequencing by tandem mass spectrometry. Rapid Commun. Mass Spectrom. **17**(20), 2337–2342 (2003)
7. Pevtsov, S., Fedulova, I., Mirzaei, H., Buck, C., Zhang, X.: Performance evaluation of existing de novo sequencing algorithms. J. Proteome Res. **5**(11), 3018–3028 (2006) ASAP Article 10.1021/pr060222h
8. Steen, H., Mann, M.: The ABC's (and XYZ's) of peptide sequencing. Nat. Rev. Mol. Cell Biol. **5**(9), 699–711 (2004)
9. Xu, C., Ma, B.: Software for Computational Peptide Identification from MS/MS. Drug Discov. Today **11**(13/14), 595–600 (2006)

Perceptron Algorithm

1959; Rosenblatt

SHAI SHALEV-SHWARTZ
Toyota Technological Institute, Chicago, IL, USA

Keywords and Synonyms

Online learning, Single layer neural network

Problem Definition

The Perceptron algorithm [1,13] is an iterative algorithm for learning classification functions. The Perceptron was mainly studied in the online learning model. As an online learning algorithm, the Perceptron observes instances in a sequence of trials. The observation at trial t is denoted by $\mathbf{x}_t$. After each observation, the Perceptron predicts a yes/no (+/−) outcome, denoted $\hat{y}_t$, which is calculated as follows

$$\hat{y}_t = \text{sign}(\langle \mathbf{w}_t, \mathbf{x}_t \rangle) ,$$

where $\mathbf{w}_t$ is a weight vector which is learned by the Perceptron and $\langle \cdot, \cdot \rangle$ is the inner product operation. Once the Perceptron has made a prediction, it receives the correct outcome, denoted y_t, where $y_t \in \{+1, -1\}$. If the prediction of the Perceptron was incorrect it updates its weight vector, presumably improving the chance of making an accurate prediction on subsequent trials. The update rule of the Perceptron is

$$\mathbf{w}_{t+1} = \begin{cases} \mathbf{w}_t + y_t \mathbf{x}_t & \text{if} \hat{y}_t \neq y_t \\ \mathbf{w}_t & \text{otherwise} \end{cases} . \quad (1)$$

The quality of an online learning algorithm is measured by the number of prediction mistakes it makes along its run. Novikoff [12] and Block [2] have shown that whenever the Perceptron is presented with a sequence of linearly separable examples, it makes a bounded number of prediction mistakes which does not depend on the length of the sequence of examples. Formally, let $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$ be a sequence of instance-label pairs. Assume that there exists a unit vector $\mathbf{u}$ ($\|\mathbf{u}\|_2 = 1$) and a positive scalar $\gamma > 0$ such that for all t, $y_t(\mathbf{u} \cdot \mathbf{x}_t) \geq \gamma$. In words, $\mathbf{u}$ separates the instance space into two half-spaces such that positively labeled instances reside in one half-space while the negatively labeled instances belong to the second half-space. Moreover, the distance of each instance to the separating hyperplane $\{\mathbf{x} : \mathbf{u} \cdot \mathbf{x} = 0\}$, is at least γ. The scalar γ is often referred to as the margin attained by $\mathbf{u}$ on the sequence of

examples. Novikoff and Block proved that the number of prediction mistakes the Perceptron makes on a sequence of linearly separable examples is at most $(R/\gamma)^2$, where $R = \max_t \|\mathbf{x}_t\|_2$ is the minimal radius of a ball enclosing all the instances. In 1969, Minsky and Papert [11] underscored serious limitations of the Perceptron by showing that it is impossible to learn many classes of patterns using the Perceptron (for example, XOR functions). This fact caused a significant decrease of interest in the Perceptron. The Perceptron has gained back its popularity after Freund and Schapire [9] proposed to use it in conjunction with kernels. The kernel-based Perceptron not only can handle non-separable datasets but can also be utilized for efficiently classifying non-vectorial instances such as trees and strings (see for example [5]).

To implement the Perceptron in conjunction with kernels one can utilize the fact that at each trial of the algorithm, the weight vector $\mathbf{w}_t$ can be written as a linear combination of the instances

$$\mathbf{w}_t = \sum_{i \in I_t} y_i \, \mathbf{x}_i \ ,$$

where $I_t = \{i < t : \hat{y}_i \neq y_i\}$ is the indices of trials in which the Perceptron made a prediction mistake. Therefore, the prediction of the algorithm can be rewritten as

$$\hat{y}_t = \text{sign}\left(\sum_{i \in I_t} y_i \ \langle \mathbf{x}_i, \mathbf{x}_t \rangle \right) \ ,$$

and the update rule of the weight vector can be replaced with an update rule for the set of erroneous trials

$$I_{t+1} = \begin{cases} I_t \cup \{t\} & \text{if } \hat{y}_t \neq y_t \\ I_t & \text{otherwise} \end{cases} \ . \qquad (2)$$

In the kernel-based Perceptron, the inner product $\langle \mathbf{x}_i, \mathbf{x}_t \rangle$ is replaced with a Mercer kernel function, $K(\mathbf{x}_i, \mathbf{x}_t)$, without any further changes to the algorithm (for a discussion on Mercer kernels see for example [15]). Intuitively, the kernel function $K(\mathbf{x}_i, \mathbf{x}_t)$ implements an inner product $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_t) \rangle$ where ϕ is a non-linear mapping from the original instance space into another (possibly high dimensional) Hilbert space. Even if the original instances are not linearly separable, the images of the instances due to the non-linear mapping ϕ can be linearly separable and thus the kernel-based Perceptron can handle non-separable datasets. Since the analysis of the Perceptron does not depend on the dimensionality of the instances, all of the formal results still hold when the algorithm is used in conjunction with kernel functions.

Key Results

In the following a mistake bound for the Perceptron in the non-separable case (see for example [10,14]) is provided.

Theorem *Assume that the Perceptron is presented with the sequence of examples* $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$ *and denote* $R = \max_t \|\mathbf{x}_t\|_2$. *Let* $\mathbf{u}$ *be a unit length weight vector* ($\|\mathbf{u}\|_2 = 1$), *let* $\gamma > 0$ *be a scalar, and denote*

$$L = \sum_{t=1}^{T} \max\{0, 1 - y_t \, \langle \mathbf{u}/\gamma, \mathbf{x}_t \rangle\} \ .$$

Then, the number of prediction mistakes the Perceptron makes on the sequence of example is at most

$$L + \left(\frac{R}{\gamma}\right)^2 + \frac{R\sqrt{L}}{\gamma} \ .$$

Note that if there exists $\mathbf{u}$ and γ such that $y_t \, \langle \mathbf{u}, \mathbf{x}_t \rangle \geq \gamma$ for all t then $L = 0$ and the above bound reduces to Novikoff's bound,

$$\left(\frac{R}{\gamma}\right)^2 \ .$$

Note also that the bound does not depend on the dimensionality of the instances. Therefore, it holds for the kernel-based Perceptron as well with $R = \max_t K(\mathbf{x}_t, \mathbf{x}_t)$.

Applications

So far the Perceptron has been viewed in the prism of online learning. Freund and Schapire [9] proposed a simple conversion of the Perceptron algorithm to the batch learning setting. A batch learning algorithm receives as input a training set of examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$ sampled independently from an underlying joint distribution over the instance and label space. The algorithm is required to output a single classification function which performs well on unseen examples as long as the unseen examples are sampled from the same distribution as the training set. The conversion of the Perceptron to the batch setting proposed by Freund and Schapire is called the voted Perceptron algorithm. The idea is to simply run the online Perceptron on the training set of examples, thus producing a sequence of weight vectors $\mathbf{w}_1, \ldots, \mathbf{w}_T$. Then, the single classification function to be used for unseen examples is a majority vote over the predictions of the weight vectors. That is,

$$f(\mathbf{x}) = \begin{cases} +1 & \text{if } |\{t : \langle \mathbf{w}_t, \mathbf{x} \rangle > 0\}| > |\{t : \langle \mathbf{w}_t, \mathbf{x} \rangle < 0\}| \\ -1 & \text{otherwise} \end{cases}$$

It was shown (see again [9]) that if the number of prediction mistakes the Perceptron makes on the training set is

Perceptron Algorithm, Table 1

Online Perceptron	Kernel-based Online Perceptron
INITIALIZATION: $\mathbf{w}_1 = \mathbf{0}$	INITIALIZATION: $\mathbf{I}_1 = \{\cdot\}$
For $t = 1, 2, \ldots$	For $t = 1, 2, \ldots$
Receive an instance $\mathbf{x}_t$	Receive an instance $\mathbf{x}_t$
Predict an outcome $\hat{y}_t = \text{sign}(\langle \mathbf{w}_t, \mathbf{x}_t \rangle)$	Predict an outcome $\hat{y}_t = \text{sign}\left(\sum_{i \in I_t} K(\mathbf{x}_i, \mathbf{x}_t)\right)$
Receive correct outcome $y_t \in \{+1, -1\}$	Receive correct outcome $y_t \in \{+1, -1\}$
Update: $\mathbf{w}_{t+1} = \begin{cases} \mathbf{w}_t + y_t \mathbf{x}_t & \text{if } \hat{y}_t \neq y_t \\ \mathbf{w}_t & \text{otherwise} \end{cases}$	Update: $I_{t+1} = \begin{cases} I_t \cup \{t\} & \text{if } \hat{y}_t \neq y_t \\ I_t & \text{otherwise} \end{cases}$

small, then $f(\mathbf{x})$ is likely to perform well on unseen examples as well.

Finally, it should be noted that the Perceptron algorithm was used for other purposes such as solving linear programming [3] and training support vector machines [14]. In addition, variants of the Perceptron was used for numerous additional problems such as online learning on a budget [8,4], multiclass categorization and ranking problems [6,7], and discriminative training for hidden Markov models [5].

Cross References

► Support Vector Machines

Recommended Reading

1. Agmon., S.: The relaxation method for linear inequalities. Can. J. Math. **6**(3), 382–392 (1954)
2. Block., H. D.: The perceptron: A model for brain functioning. Rev. Mod. Phys. **34**, 123–135 (1962)
3. Blum, A., Dunagan J. D.: Smoothed analysis of the perceptron algorithm for linear programming. In: SODA, (2002)
4. Cesa-Bianchi, N., Gentile, C.: Tracking the best hyperplane with a simple budget perceptron. In: Proceedings of the Nineteenth Annual Conference on Computational Learning Theory, (2006)
5. Collins, M.: Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In: Conference on Empirical Methods in Natural Language Processing, (2002)
6. Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., Singer, Y.: Online passive aggressive algorithms. J. Mach. Learn. Res. **7** (2006)
7. Crammer, K., Singer, Y.: A new family of online algorithms for category ranking. In: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (2002)
8. Dekel, O., Shalev-Shwartz, S., Singer, Y.: The Forgetron: A kernel-based perceptron on a fixed budget. In: Advances in Neural Information Processing Systems 18 (2005)
9. Freund, Y., Schapire, R. E.: Large margin classification using the perceptron algorithm. In: Proceedings of the Eleventh Annual Conference on Computational Learning Theory (1998)
10. Gentile, C.: The robustness of the p-norm algorithms. Mach. Learn. **53**(3) (2002)
11. Minsky, M., Papert, S.: Perceptrons: An Introduction to Computational Geometry. The MIT Press, (1969)
12. Novikoff, A. B. J.: On convergence proofs on perceptrons. In: Proceedings of the Symposium on the Mathematical Theory of Automata, volume XII, pp. 615–622, (1962)
13. Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**, 386–407 (1958)
14. Shalev-Shwartz, S., Singer, Y.: A new perspective on an old perceptron algorithm. In: Proceedings of the Eighteenth Annual Conference on Computational Learning Theory, (2005)
15. Vapnik, V. N.: Statistical Learning Theory. Wiley (1998)

Perfect Phylogeny (Bounded Number of States)

1997; Kannan, Warnow

JESPER JANSSON
Ochanomizu University, Tokyo, Japan

Keywords and Synonyms

Compatibility of characters with a bounded number of states; Convex tree-realization of partitions containing a bounded number of sets

Problem Definition

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of elements called *objects* and *species*, and let $C = \{c_1, c_2, \ldots, c_m\}$ be a set of functions called *characters* such that each $c_j \in C$ is a function from S to the set $\{0, 1, \ldots, r_j - 1\}$ for some integer r_j. For every $c_j \in C$, the set $\{0, 1, \ldots, r_j - 1\}$ is called the set of *allowed states* of character c_j, and for any $s_i \in S$ and $c_j \in C$, it is said that the *state of* s_i *on* c_j is α, or that the *state of* c_j *for* s_i is α, where $\alpha = c_j(s_i)$. The *character state matrix* for S and C is the $(n \times m)$-matrix in which entry (i, j) for any

$i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, m\}$ equals the state of s_i on c_j.

In this chapter, a *phylogeny for* S is an unrooted tree whose leaves are bijectively labeled by S. For every $c_j \in C$ and $\alpha \in \{0, 1, \ldots, r_j - 1\}$, define the set $S_{c_j,\alpha}$ by $S_{c_j,\alpha} = \{s_i \in S :$ the state of s_i on c_j is $\alpha\}$. A *perfect phylogeny for* (S, C) (if one exists) is a phylogeny T for S such that the following holds: for each $c_j \in C$ and pair of allowed states α, β of c_j with $\alpha \neq \beta$, the minimal subtree of T that connects $S_{c_j,\alpha}$ and the minimal subtree of T that connects $S_{c_j,\beta}$ are vertex-disjoint. See Fig. 1 for an example. *The Perfect Phylogeny Problem* is the following:

Problem 1 (The Perfect Phylogeny Problem)
INPUT: *A character state matrix M for some S and C.*
OUTPUT: *A perfect phylogeny for (S, C), if one exists; otherwise, null.*

Below, define $r = \max_{j \in \{1,2,\ldots,m\}} r_j$.

Key Results

The following negative result was proved by Bodlaender, Fellows, and Warnow [2] and, independently, by Steel [13]:

Theorem 1 ([2,13]) *The Perfect Phylogeny Problem is NP-hard.*

On the other hand, certain restrictions of The Perfect Phylogeny Problem can be solved efficiently. One important special case occurs if the number of allowed states of each character is limited[1]. For this case, Agarwala and Fernández-Baca [1] designed a dynamic programming-based algorithm that builds perfect phylogenies on certain subsets of S called *c-clusters* (also referred to as *proper clusters* in [5,10] and *character subfamilies* in [6]) in a bottom-up fashion. Each c-cluster G has the property that: (1) G and $S \setminus G$ share at most one state of each character; and (2) for at least one character, G and $S \setminus G$ share no states. The number of c-clusters is at most $2^r m$, and the algorithm's total running time is $O(2^{3r}(nm^3 + m^4))$, i. e., exponential in r. (Hence, The Perfect Phylogeny Problem is polynomial-time solvable if the number of allowed states of every character is upper-bounded by $O(\log(m + n))$.) Subsequently, Kannan and Warnow [10] presented a modified algorithm with improved running time. They restructured the algorithm of [1] to eliminate one of the three nested loops that steps through all possible c-clusters and added a pre-processing step which speeds up the innermost loop. The resulting time complexity is given by:

Perfect Phylogeny (Bounded Number of States), Table 1
The running times of the fastest known algorithms for The Perfect Phylogeny Problem with a bounded number of states

r	Running time	Reference
2	$O(nm)$	[11] together with [7]
3	$\min\{O(nm^2), O(n^2 m)\}$	[3,10] together with [9]
4	$\min\{O(nm^2), O(n^2 m)\}$	[10] together with [9]
≥ 5	$O(2^{2r} nm^2)$	[10]

Theorem 2 ([10]) *The algorithm of Kannan and Warnow in [10] solves The Perfect Phylogeny Problem in* $O(2^{2r} nm^2)$ *time.*

A perfect phylogeny T for (S, C) is called *minimal* if no tree which results by contracting an edge of T is a perfect phylogeny for (S, C). In [10], Kannan and Warnow also showed how to extend their algorithm to enumerate all minimal perfect phylogenies for (S, C) by constructing a directed acyclic graph that implicitly stores the set of all perfect phylogenies for (S, C).

Theorem 3 ([10]) *The extended algorithm of Kannan and Warnow in [10] enumerates the set of all minimal perfect phylogenies for (S, C) so that the maximum computation time between two consecutive outputs is* $O(2^{2r} nm^2)$.

For very small values of r, even faster algorithms are known. Refer to the table in Table 1 for a summary. If $r = 2$ then the problem can be solved in $O(nm)$ time by reducing it to The *Directed* Perfect Phylogeny Problem for Binary Characters (see, e. g., Encyclopedia ▶ Directed Perfect Phylogeny (Binary Characters) for a definition of this variant of the problem) using $O(nm)$ time [7,11] and then applying Gusfield's $O(nm)$-time algorithm [7]. If $r = 3$ or $r = 4$, the problem is solvable in $O(n^2 m)$ time by another algorithm by Kannan and Warnow [9], which is faster than the algorithm from Theorem 2 when $n < m$. Also note that for the case $r = 3$, there exists an older algorithm by Dress and Steel [3] whose running time coincides with that of the algorithm in Theorem 2.

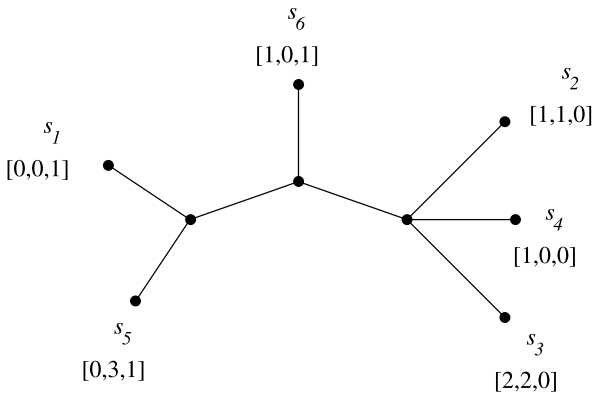
Applications

A central goal in computational evolutionary biology and phylogenetic reconstruction is to develop efficient methods for constructing, from some given data, a phylogenetic tree that accurately describes the evolutionary relationships among a set of objects (e. g., biological species or

[1] For other variants of The Perfect Phylogeny Problem which can be solved efficiently, see, for example, entries ▶ Directed Perfect Phylogeny (Binary Characters) of this Encyclopedia or the survey by Fernández-Baca [5] .

M	c_1	c_2	c_3
s_1	0	0	1
s_2	1	1	0
s_3	2	2	0
s_4	1	0	0
s_5	0	3	1
s_6	1	0	1

(a)



(b)

Perfect Phylogeny (Bounded Number of States), Figure 1
a An example of a character state matrix M for $S = \{s_1, s_2, \ldots, s_6\}$ and $C = \{c_1, c_2, c_3\}$ with $r_1 = 3$, $r_2 = 4$, and $r_3 = 2$, i. e., $r = 4$. b A perfect phylogeny for (S, C). For convenience, the states of all three characters for each object are shown

other taxa, populations, proteins, genes, natural languages, etc.) believed to have been produced by an evolutionary process. One of the most widely used techniques for reconstructing a phylogenetic tree is to represent the objects as vectors of character states and look for a tree that clusters objects which have a lot in common. The Perfect Phylogeny Problem can be regarded as the ideal special case of this approach in which the given data contains no errors, evolution is tree-like, and each character state can emerge only once in the evolutionary history.

However, data obtained experimentally seldom admits a perfect phylogeny, so various optimization versions of the problem such as *maximum parsimony* and *maximum compatibility* are often considered in practice; as might be expected, these strategies generally lead to NP-complete problems, but there exist many heuristics that work well for most inputs. See, e. g. [4,5,12], for a further discussion and references. Nevertheless, algorithms for The Perfect Phylogeny Problem may be useful even when the data does not admit a perfect phylogeny, for example if there exists a perfect phylogeny for $m - O(1)$ of the characters in C. In fact, in one crucial step of their proposed character-based methodology for determining the evolutionary history of a set of related natural languages, Warnow, Ringe, and Taylor [14] consider all subsets of C in decreasing order of cardinality, repeatedly applying the algorithm of [10] until a largest subset of C which admits a perfect phylogeny is found. The ideas behind the algorithms of [1] and [10] have also been utilized and extended by Fernández-Baca and Lagergren [6] in their algorithm for computing *near-perfect phylogenies* in which the constraints on the output have been relaxed in order to permit non-perfect phylogenies whose so-called penalty score is less than or equal to a prespecified parameter q (see [6] for details).

The motivation for considering a bounded number of states is that characters based on directly observable traits are, by the way they are defined, naturally bounded by some small number (often 2). When biomolecular data is used to define characters, the number of allowed states is typically bounded by a constant; e. g., $r = 2$ for SNP markers, $r = 4$ for DNA or RNA sequences, or $r = 20$ for amino-acid sequences (see also Encyclopedia ► Directed Perfect Phylogeny (Binary Characters)). Moreover, characters with $r = 2$ can be useful in comparative linguistics [8].

Open Problems

An important open problem is to determine whether the running time of the algorithm of Kannan and Warnow [10] can be improved. As pointed out in [5], it would be especially interesting to find out if The Perfect Phylogeny Problem is solvable in $O(2^{2r}nm)$ time for any r, or more generally, in $O(f(r)\cdot nm)$ time, where f is a function of r which does not depend on n or m, since this would match the fastest known algorithm for the special case $r = 2$ (see Table 1). Another open problem is to establish lower bounds on the computational complexity of The Perfect Phylogeny Problem with a bounded number of states.

Cross References

► Directed Perfect Phylogeny (Binary Characters)
► Perfect Phylogeny Haplotyping

Acknowledgments

Supported in part by Kyushu University, JSPS (Japan Society for the Promotion of Science), and INRIA Lille – Nord Europe.

Recommended Reading

1. Agarwala, R., Fernández-Baca, D.: A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. SIAM J. Comput. **23**, 1216–1224 (1994)
2. Bodlaender, H.L., Fellows, M.R., Warnow, T.: Two strikes against perfect phylogeny. In: Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP 1992). Lecture Notes in Computer Science, vol. 623, pp. 273–283. Springer, Berlin (1992)
3. Dress, A., Steel, M.: Convex tree realizations of partitions. Appl. Math. Lett. **5**, 3–6 (1992)
4. Felsenstein, J.: Inferring Phylogenies. Sinauer Associates, Inc. Sunderland, Massachusetts (2004)
5. Fernández-Baca, D.: The Perfect Phylogeny Problem. In: Cheng, X., Du D.-Z. (eds.) Steiner Trees in Industry, pp. 203–234. Kluwer Academic Publishers, Dordrecht, Netherlands (2001)
6. Fernández-Baca, D., Lagergren, J.: A polynomial-time algorithm for near-perfect phylogeny. SIAM J. Comput. **32**, 1115–1127 (2003)
7. Gusfield, D.M.: Efficient algorithms for inferring evolutionary trees. Networks **21**, 19–28 (1991)
8. Kanj, I.A., Nakhleh, L., Xia, G.: Reconstructing evolution of natural languages: Complexity and parametrized algorithms. In: Proceedings of the 12th Annual International Computing and Combinatorics Conference (COCOON 2006). Lecture Notes in Computer Science, vol. 4112, pp. 299–308. Springer, Berlin (2006)
9. Kannan, S., Warnow, T.: Inferring evolutionary history from DNA sequences. SIAM J. Comput. **23**, 713–737 (1994)
10. Kannan, S., Warnow, T.: A fast algorithm for the computation and enumeration of perfect phylogenies. SIAM J. Comput. **26**, 1749–1763 (1997)
11. McMorris, F.R.: On the compatibility of binary qualitative taxonomic characters. Bull. Math. Biol. **39**, 133–138 (1977)
12. Setubal, J.C., Meidanis, J.: Introduction to Computational Molecular Biology. PWS Publishing Company, Boston (1997)
13. Steel, M.A.: The complexity of reconstructing trees from qualitative characters and subtrees. J. Classification **9**, 91–116 (1992)
14. Warnow, T., Ringe, D., Taylor, A.: Reconstructing the evolutionary history of natural languages. In: Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'96), pp. 314–322 (1996)

Perfect Phylogeny Haplotyping

2005; Ding, Filkov, Gusfield

Giuseppe Lancia
Department of Mathematics and Computer Science, University of Udine, Udine, Italy

Keywords and Synonyms

Alleles phasing

Problem Definition

In the context of the *perfect phylogeny haplotyping* (PPH) problem, each vector $h \in \{0,1\}^m$ is called a *haplotype*, while each vector $g \in \{0,1,2\}^m$ is called a *genotype*. Haplotypes are binary encodings of DNA sequences, while genotypes are ternary encodings of *pairs* of DNA sequences (one sequence for each of the two homologous copies of a certain chromosome).

Two haplotypes h' and h'' are said to *resolve* a genotype g if, at each position j: (i) if $g_j \in \{0,1\}$ then both $h'_j = g_j$ and $h''_j = g_j$; (ii) if $g_j = 2$ then either $h'_j = 0$ and $h''_j = 1$ or $h'_j = 1$ and $h''_j = 0$. If h' and h'' resolve g, we write $g = h' + h''$. An instance of the PPH problem consists of a set $G = \{g^1, g^2, \ldots, g^n\}$ of genotypes. A set H of haplotypes such that, for each $g \in G$, there are $h', h'' \in H$ with $g = h' + h''$, is called a *resolving set* for G.

A *perfect phylogeny* for a set H of haplotypes is a rooted tree T for which

- the set of leaves is H and the root is labeled by some binary vector r;
- each index $j \in \{1, \ldots, m\}$ labels exactly one edge of T;
- if an edge e is labeled by an index k, then, for each leaf h that can be reached from the root via a path through e, it is $h_k \neq r_k$.

Without loss of generality, it can be assumed that the vector labeling the root is $r = 0$. Within the PPH problem, T is meant to represent the evolution of the sequences at the leaves from a common ancestral sequence (the root). Each edge labeled with an index represents a point in time when a mutation happened at a specific site. This model of evolution is also known as *coalescent* [11]. It can be shown that a perfect phylogeny for H exists if and only if for all choices of four haplotypes $h^1, \ldots, h^4 \in H$ and two indices i, j,

$$\{h^a_i\, h^a_j\,, 1 \le a \le 4\} \neq \{00, 01, 10, 11\}\,.$$

Given the above definitions, the problem surveyed in this entry is the following:

Perfect Phylogeny Haplotyping Problem (PPH): Given a set G of genotypes, find a resolving set H of haplotypes and a perfect phylogeny T for H, or determine that such a resolving set does not exist.

In a slightly different version of the above problem, one may require to find all perfect phylogenies for H instead of just one (in fact, all known algorithms for PPH do find all perfect phylogenies).

The perfect phylogeny problem was introduced by Gusfield [7], who also proposed a nearly linear-time $O(nm\,\alpha(nm))$-algorithm for its solution (where $\alpha()$ is the extremely slowly growing inverse Ackerman function).

The algorithm resorted to a reduction to a complex procedure for the *graph realization* problem (Bixby and Wagner [2]), of very difficult understanding and implementation. Later approaches for PPH proposed much simpler, albeit slower, $O(nm^2)$-algorithms (Bafna et al. [1]; Eskin et al. [6]). However, a major question was left open: *does there exist a linear-time algorithm for PPH*? In [7], Gusfield conjectured that this should be the case. The 2005 algorithm by Ding, Filkov, and Gusfield [5] surveyed in this entry settles the above conjecture in the affirmative.

Key Results

The main idea of the algorithm is to find the maximal sub-graphs that are common to all PPH solutions. Let us call *P-class* a maximal sub-graph of all PPH trees for G. The authors show that each P-class consists of two sub-trees which, in each PPH solution, can appear in either one of two possible ways (called *flips* of the P-class) with respect to any fixed P-class taken as a reference. Hence, if there are k P-classes, there are 2^{k-1} distinct PPH solutions.

The algorithm resorts to an original and effective data structure, called the *shadow tree*, which gives an implicit representation of all P-classes. The data structure is built incrementally, by processing one genotype at a time. The total cost for building and updating the shadow tree is linear in the input size (i. e., in nm). A detailed description of the shadow tree requires a rather large number of definitions, possibly accompanied by figures and examples. Here, we will introduce only its basic features, those that allow us to state the main theorems of [5].

The shadow tree is a particular type of directed rooted tree, which contains both *edges* and *links* (strictly speaking, the latter are just arcs, but they are called links to underline their specific use in the algorithm). The edges are of two types: *tree-edges* and *shadow-edges*, and are associated to the indices $\{1, \ldots, m\}$. For each index i, there is a tree-edge labeled t_i and a shadow-edge labeled s_i. Both edges and links are oriented, with their head closer to the root than their tail. Other than the root, each node of the shadow tree is the endpoint of exactly one tree-edge or one shadow-edge (while the root is the head of two "dummy" links). The links are used to connect certain tree- and shadow-edges. A link can be either *free* or *fixed*. The head of a free link can still be changed during the execution of the algorithm, but once a link is fixed, it cannot be changed any more.

Tree-edges, shadow-edges and *fixed* links are organized into *classes*, which are sub-graphs of the shadow tree. Each fixed link is contained in exactly one class, while each free link connects one class to another, called its *parent*. For each index i, if the tree-edge t_i is in a class X, then the shadow-edge s_i is in X as well, so that a class can be seen as a pair of "twin" sub-trees of the shadow tree. The free links point out from the root of the sub-trees (the *class roots*). Classes change during the running of the algorithm. Specifically, classes are *created* (containing a single tree- and shadow-edge) when a new genotype is processed; a class can be *merged* with its parent, by fixing a pair of free edges; finally, a class can be *flipped*, by switching the heads of the two free links that connect the class roots to the parent class.

A tree T is said to be "*contained in*" a shadow tree if T can be obtained by flipping some classes in the shadow tree, followed by contracting all links and shadow-edges. Let us call *contraction* of a class the sub-graph (consisting of a pair of sub-trees, made of tree-edges only) that is obtained from a class X of the shadow tree by contracting all fixed links and shadow-edges of X. The following are the main results obtained in [5]:

Proposition 1 *Every P-class can be obtained by contraction of a class of the final shadow tree produced by the algorithm. Conversely, every contraction of a class of the final shadow tree is a P-class.*

Theorem 1 *Every PPH solution is contained in the final shadow tree produced by the algorithm. Conversely, every tree contained in the final shadow tree is a distinct PPH solution.*

Theorem 2 *The total time required for building and updating the shadow tree is $O(nm)$.*

Applications

The PPH problem arises in the context of *Single Nucleotide Polymorphisms* (SNP's) analysis in human genomes. A SNP is the site of a single nucleotide which varies in a statistically significant way in a population. The determination of SNP locations and of common SNP patterns (haplotypes) are of uttermost importance. In fact, SNP analysis is used to understand the nature of several genetic diseases, and the international Haplotype Map Project is devoted to SNP study (Helmuth [9]).

The values that a SNP can take are called its *alleles*. Almost all SNPs are bi-allelic, i. e., out of the four nucleotides A, C, T, G, only two are observed at any SNP. Humans are *diploid* organisms, with DNA organized in pairs of chromosomes (of paternal and of maternal origin). The sequence of alleles on a chromosome copy is called a *haplotype*. Since SNPs are bi-allelic, haplotypes can be encoded as binary strings. For a given SNP, an individual can be either *homozygous*, if both parents contributed the same allele, or *heterozygous*, if the paternal and maternal alleles are different.

Haplotyping an individual consists of determining his two haplotypes. Haplotyping a population consists of haplotyping each individual of the population. While it is today economically infeasible to determine the haplotypes directly, there is a cheap experiment which can determine the (less informative and often ambiguous) *genotypes*.

A genotype of an individual contains the *conflated* information about the two haplotypes. For each SNP, the genotype specifies which are the two (possibly identical) alleles, but does not specify their origin (paternal or maternal). The ternary encoding that is used to represent a genotype g has the following meaning: at each SNP j, it is $g_j = 0$ (respectively, 1) if the individual is homozygous for the allele 0 (respectively, 1), and $g_j = 2$ if the individual is heterozygous. There may be many possible pairs of haplotypes that justify a particular genotype (there are 2^{k-1} pairs of haplotypes that can resolve a genotype with k heterozygous SNPs). Given a set of genotypes, in order to determine the correct resolving set out of the exponentially many possibilities, one imposes some "biologically meaningful" constraints that the solution must possess. The perfect phylogeny model (coalescent) requires that the resolving set must fit a particular type of evolutionary tree. That is, all haplotypes should descend from some ancestral haplotype, via mutations that happened (only once) at specific sites over time. The coalescent model is accurate especially for short haplotypes (for longer haplotypes there is also another type of evolutionary event, *recombination*, that should be taken into account).

The linear-time PPH algorithm is of significant practical value in two respects. First, there are instances of the problem where the number of SNPs considered is fairly large (genotypes can extend over several kilo-bases). For these long instances, the advantage of an $O(nm)$ algorithm with respect to the previous $O(nm^2)$ approach is evident. On the other hand, when genotypes are relatively short, the benefit of using the linear-time algorithm is not immediately evident (both algorithms run extremely quickly). Nevertheless, there are situations in which one has to solve a large set of haplotyping problems, where each single problem is defined over short genotypes. For instance, this is the case in which one examines all "small" subsets of SNPs in order to determine the subsets for which there is a PPH solution. In this type of application, the gain of efficiency with the use of the linear-time PPH algorithm is significant (Chung and Gusfield [4]; Wiuf [14]).

Open Problems

A linear-time algorithm is the best possible for PPH, and no open problems are listed in [5].

Experimental Results

The algorithm has been implemented in C and its performance has been compared with the previous fastest PPH algorithm, i. e. DPPH (Bafna et al. [1]). In the case of $m = 2000$ and $n = 1000$, the algorithm is about 250-times faster than DPPH, and is capable of solving an instance in an average time of 2 seconds, versus almost 8 minutes needed by DPPH (on a "standard" 2005 Personal Computer). The smaller instances (e. g., with $m = 50$ SNPs) are such that the superior performance of the algorithm is not as evident, with an average running time of 0.07 seconds versus 0.2 seconds. However, as already remarked, when the small instances are executed within a loop, the speed-up turns out to be again of two or more orders of magnitude.

Data Sets

The data sets used in [5] have been generated by the program *ms* (Hudson [12]), which is the widely used standard for instance generation reflecting the coalescent model of SNP sequence evolution. Real-life instances can be found at the HapMap web site http://www.hapmap.org.

URL to Code

http://wwwcsif.cs.ucdavis.edu/~gusfield/lpph/

Cross References

- ▶ Directed Perfect Phylogeny (Binary Characters)
- ▶ Perfect Phylogeny (Bounded Number of States)

Recommended Reading

For surveys about computational haplotyping problems in general, see Bonizzoni et al. [3], Gusfield and Orzack [8], Halldorsson et al. [10], and Lancia [13].

1. Bafna, V., Gusfield, D., Lancia, G., Yooseph, S.: Haplotyping as perfect phylogeny: a direct approach. J. Comput. Biol. **10**(3–4), 323–340 (2003)
2. Bixby, R.E., Wagner, D.K.: An almost linear-time algorithm for graph realization. Math. Oper. Res. **13**, 99–123 (1988)
3. Bonizzoni, P., Della Vedova, G., Dondi, R., Li, J.: The haplotyping problem: an overview of computational models and solutions. J. Comput. Sci. Technol. **19**(1), 1–23 (2004)
4. Chung, R.H., Gusfield, D.: Empirical exploration of perfect phylogeny haplotyping and haplotypes. In: Proceedings of Annual International Conference on Computing and Combinatorics (COCOON). Lecture Notes in Computer Science, vol. 2697, pp. 5–9. Springer, Berlin (2003)
5. Ding, Z., Filkov, V., Gusfield, D.: A linear-time algorithm for the perfect phylogeny haplotyping problem. In: Proceedings of the Annual International Conference on Computational

Molecular Biology (RECOMB), New York, 2005. ACM Press, New York (2005)
6. Eskin, E., Halperin, E., Karp, R.: Efficient reconstruction of haplotype structure via perfect phylogeny. J. Bioinform. Comput. Biol. **1**(1), 1–20 (2003)
7. Gusfield, D.: Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions. In: Myers, G., Hannenhalli, S., Istrail, S., Pevzner, P., Waterman, M. (eds.) Proceedings of the Annual International Conference on Computational Molecular Biology (RECOMB), New York, 2002, pp. 166–175. ACM Press (2002)
8. Gusfield, D. Orzack, S.H.: Haplotype inference. In: Aluru S. (ed) Handbook of Computational Molecular Biology, pp. 1–28. Champman and Hall/CRC-press, Boca Raton (2005)
9. Helmuth, L.: Genome research: Map of the human genome 3.0. Science **293**(5530), 583–585 (2001)
10. Halldorsson, B.V., Bafna, V., Edwards, N., Lippert, R., Yooseph, S., Istrail, S.: A survey of computational methods for determining haplotypes. In: Computational methods for SNP and haplotype inference: DIMACS/RECOMB satellite workshop. Lecture Notes in Computer Science, vol. 2983, pp. 26–47. Springer, Berlin (2004)
11. Hudson, R.: Gene genealogies and the coalescent process. Oxf. Surv. Evol. Biol. **7**, 1–44 (1990)
12. Hudson, R.: Generating samples under the wright-fisher neutral model of genetic variation. Bioinformatics **18**(2), 337–338 (2002)
13. Lancia, G.: The phasing of heterozygous traits: Algorithms and complexity. Comput. Math. Appl. **55**(5), 960–969 (2008)
14. Wiuf, C.: Inference on recombination and block structure using unphased data. Genetics **166**(1), 537–545 (2004)

Performance Analysis

► Distance-Based Phylogeny Reconstruction (Optimal Radius)

Performance-Driven Clustering

1993; Rajaraman, Wong

Rajmohan Rajaraman
Department of Computer Science,
Northeastern University,
Boston, MA, USA

Keywords and Synonyms

Circuit partitioning; Circuit clustering

Problem Definition

Circuit partitioning consists of dividing the circuit into parts each of which can be implemented as a separate component (e. g., a chip), that satisfies the design constraints. The work of Rajaraman and Wong [5] considers the problem of dividing a circuit into components, subject to area constraints, such that the maximum delay at the outputs is minimized.

A combinational circuit can be represented as a directed acyclic graph $G = (V, E)$, where V is the set of nodes, and E is the set of directed edges. Each node represents a gate in the network and each edge (u, v) in E represents an interconnection between gates u and v in the network. The *fanin* of a node is the number of edges incident into it, and the *fanout* of of a node is the number of edges incident out of it. A *primary input (PI)* is a node with fanin 0, while a *primary output (PO)* is a node with fanout 0. Each node has a *weight* and a *delay* associated with it.

Definition 1 A clustering of a network $G = (V, E)$ is a triple (H, ϕ, Σ), where

1. $H = (V', E')$ is a directed acyclic graph.
2. φ is a function mapping V' to V such that
   - For every edge $(u', v') \in E'$, $(\phi(u'), \phi(v')) \in E$.
   - For every node $v' \in V'$ and edge $(u, \phi(v')) \in E$, there exists a unique $u' \in V'$ such that $\phi(u') = u$ and $(u', v') \in E'$.
   - For every PO node $v \in V$, there exists a unique $v' \in V'$ such that $\phi(v') = v$.
3. Σ is a partition of V'.

Let $\Gamma = (H = (V', E'), \phi, \Sigma)$ be a clustering of G. For $v \in V$, $v' \in V'$, if $\phi(v') = v$, we call v' a *copy* of v. The set V' consists of all the copies of the nodes in V that appear in the clustering. A node v' is a PI (respectively, PO) in Γ if $\phi(v')$ is a PI (respectively, PO) in G. It follows from the definition of φ that H is logically equivalent to G.

The weights and delays on the individual nodes in G yield weights and delays of nodes in H' and a delay for the clustering Γ. The weight (respectively, delay) of an node v' in V' is the weight (respectively, delay) of $\phi(v)$. The weight of any cluster $C \in \Sigma$, denoted by $W(C)$, is the sum of the weights of the nodes in C. The delay of a clustering is given by the general delay model of Murgai et al. [3], which is as follows. The delay of an edge $(u', v') \in E'$ is D (which is a given parameter) if u' and v' belong to different elements of Σ and zero otherwise. The delay along a path in H' is simply the sum of the delays of the edges of the path. Finally, the delay of Γ is the delay of a maximum-delay path in H', among all the paths from a PI node to a PO node in H'.

Definition 2 Given a combinational network $G = (V, E)$ with weight function $w: V \to R^+$, weight capacity M and a delay function $\delta: V \to R^+$, we say that a clustering $\Gamma = (H, \phi, \Sigma)$ is *feasible* if for every cluster $C \in \Sigma$, $W(C)$

is at most M. The *circuit clustering problem* is to compute a feasible clustering Γ of G such that the delay of Γ is minimum among all feasible clusterings of G.

An early work of Lawler et al. [2] presented a polynomial-time optimal algorithm for the circuit clustering problem in the special case where all the gate delays are zero (i. e., $\delta(v) = 0$ for all v).

Key Results

Rajaraman and Wong [5] presented an optimal polynomial-time algorithm for the circuit clustering problem under the general delay model.

Theorem 1 *There exists an algorithm that computes an optimal clustering for the circuit clustering problem in $O(n^2 \log n + nm)$ time, where n and m are the vertices and edges, respectively, of the given combinational network.*

This result can be extended to compute optimal clusterings under any monotone clustering constraint. A clustering constraint is monotone if any connected subset of nodes in a feasible cluster is also monotone [2].

Theorem 2 *The circuit clustering problem can be solved optimally under any monotone clustering constraint in time polynomial in the size of the circuit.*

Applications

Circuit partitioning/clustering is an important component of very large scale integration design. One application of the circuit clustering problem formulated above is to implement a circuit on multiple field programmable gate array chips. The work of Rajaraman and Wong focused on clustering combinational circuits to minimize delay under area constraints. Related studies have considered other important constraints, such as pin constraints [1] and a combination of area and pin constraints [6]. Further work has also included clustering sequential circuits (as opposed to combinational circuits) with the objective of minimizing the clock period [4].

Experimental Results

Rajaraman and Wong reported experimental results on five ISCAS (International Symposium on Circuits and Systems) circuits. The number of nodes in these circuits ranged from 196 to 913. They reported the maximum delay of the clusterings and running times of their algorithm on a Sun Sparc workstation.

Cross References

► FPGA Technology Mapping

Recommended Reading

1. Cong, J., Ding, Y.: An optimal technology mapping algorithm for delay optimization in lookup-table based fpga design. In: Proceedings of IEEE International Conference on Computer-Aided Design, 1992, pp. 48–53
2. Lawler, E.L., Levitt, K.N., Turner, J.: Module clustering to minimize delay in digital networks. IEEE Trans. Comput. **C-18**, 47–57 (1966)
3. Murgai, R., Brayton, R.K., Sangiovanni-Vincentelli, A.: On clustering for minimum delay/area. In: Proceedings of IEEE International Conference on Computer-Aided Design, 1991, pp. 6–9
4. Pan, P., Karandikar, A.K., Liu, C.L.: Optimal clock period clustering for sequential circuits with retiming. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **17**, 489–498 (1998)
5. Rajaraman, R., Wong, D.F.: Optimum clustering for delay minimization. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **14**, 1490–1495 (1995)
6. Yang, H.H., Wong, D.F.: Circuit clustering for delay minimization under area and pinconstraints. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **16**, 976–986 (1997)

Phylogenetic Tree Construction from a Distance Matrix

1989; Hein

Jesper Jansson
Ochanomizu University, Tokyo, Japan

Keywords and Synonyms

Phylogenetic tree construction from a dissimilarity matrix

Problem Definition

Let n be a positive integer. A *distance matrix of order n* (also called a *dissimilarity matrix of order n*) is a matrix D of size $(n \times n)$ which satisfies: (1) $D_{i,j} > 0$ for all $i, j \in \{1, 2, \ldots, n\}$ with $i \neq j$; (2) $D_{i,j} = 0$ for all $i, j \in \{1, 2, \ldots, n\}$ with $i = j$; and (3) $D_{i,j} = D_{j,i}$ for all $i, j \in \{1, 2, \ldots, n\}$.

Below, all trees are assumed to be unrooted and edge-weighted. For any tree $\mathcal{T}$, the *distance* between two nodes u and v in $\mathcal{T}$ is defined as the sum of the weights of all edges on the unique path in $\mathcal{T}$ between u and v, and is denoted by $d^{\mathcal{T}}_{u,v}$. A tree $\mathcal{T}$ is said to *realize* a given distance matrix D of order n if and only if it holds that $\{1, 2, \ldots, n\}$ is a subset of the nodes of $\mathcal{T}$ and $d^{\mathcal{T}}_{i,j} = D_{i,j}$ for all $i, j \in \{1, 2, \ldots, n\}$. Finally, a distance matrix D is

called *additive* or *tree-realizable* if and only if there exists a tree which realizes D.

Problem 1 (The Phylogenetic Tree from Distance Matrix Problem)
INPUT: *An distance matrix D of order n.*
OUTPUT: *A tree which realizes D and has the smallest possible number of nodes, if D is additive; otherwise, null.*

See Fig. 1 for an example.

In the time complexities listed below, the time needed to input all of D is not included. Instead, $O(1)$ is charged to the running time whenever an algorithm requests to know the value of any specified entry of D.

Key Results

Several authors have independently shown how to solve The Phylogenetic Tree from Distance Matrix Problem. The fastest of these algorithms run in $O(n^2)$ time[1]:

Theorem 1 ([2,4,5,7,15]) *There exists an algorithm which solves The Phylogenetic Tree from Distance Matrix Problem in $O(n^2)$ time.*

Although the algorithms are different, it can be proved that:

Theorem 2 ([8,15]) *For any given distance matrix, the solution to The Phylogenetic Tree from Distance Matrix Problem is unique.*

Furthermore, the algorithms referred to in Theorem 1 have optimal running time since any algorithm for The Phylogenetic Tree from Distance Matrix Problem must in the worst case query all $\Omega(n^2)$ entries of D to make sure that D is additive. However, if it is known in advance that the input distance matrix is additive then the time complexity improves, as shown by Hein [9]:

Theorem 3 ([9,12]) *There exists an algorithm which solves The Phylogenetic Tree from Distance Matrix Problem restricted to additive distance matrices in $O(kn \log_k n)$ time, where k is the maximum degree of the tree that realizes the input distance matrix[2].*

The algorithm of Hein [9] starts with a tree containing just two nodes and then successively inserts each node i into the tree by repeatedly choosing a pair of existing nodes and computing where on the path between them that i should be attached, until i's position has been determined. (The same basic technique is used in the $O(n^2)$-time algorithm of Waterman et al. [15] referenced to by Theorem 1 above, but the algorithm of Hein selects paths which are more efficient at discriminating between the possible positions for i.)

The lower bound corresponding to Theorem 3 is given by:

Theorem 4 ([10]) *The Phylogenetic Tree from Distance Matrix Problem restricted to additive distance matrices requires $\Omega(kn \log_k n)$ queries to the distance matrix D, where k is the maximum degree of the tree that realizes D, even if restricted to trees in which all edge weights are equal to 1.*

Finally, note that the following special case is easily solvable in linear time:

Theorem 5 ([5]) *There exists an $O(n)$-time algorithm which solves The Phylogenetic Tree from Distance Matrix Problem restricted to additive distance matrices for which the realizing tree contains two leaves only and has all edge weights equal to 1.*

Applications

The main application of The Phylogenetic Tree from Distance Matrix Problem is in the construction of a tree (a so-called *phylogenetic tree*) that represents evolutionary relationships among a set of studied objects (e. g., species or other taxa, populations, proteins, genes, etc.). Here, it is assumed that the objects are indeed related according to a tree-like branching pattern caused by an evolutionary process and that their true pairwise evolutionary distances are proportional to the measured pairwise dissimilarities. See, e. g., [1,6,7,14,15] for examples and many references as well as discussions on how to estimate pairwise dissimilarities based on biological data. Other applications of The Phylogenetic Tree from Distance Matrix Problem can be found in psychology, for example to describe semantic memory organization [1], in comparative linguistics to infer the evolutionary history of a set of languages [11], or in the study of the filiation of manuscripts to trace how manuscript copies of a text (whose original version may have been lost) have evolved in order to identify discrepancies among them or to reconstruct the original text [1,3,13].

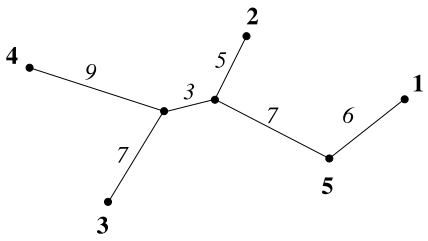
In general, real data seldom forms additive distance matrices [15]. Therefore, in practice, researchers consider optimization versions of The Phylogenetic Tree from Distance Matrix Problem which look for a tree that "almost" realizes D. Many alternative definitions of "almost" have been proposed, and numerous heuristics and approxima-

[1] See [5] for a short survey of older algorithms which do not run in $O(n^2)$ time.

[2] For this case, the Culberson-Rudnicki algorithm [5] runs in $O(n^{3/2}\sqrt{k})$ time for trees in which all edge weights are equal to 1, and not in $O(kn \log_k n)$ time as claimed in [5]. See [12] for a counterexample to [5] and a correct analysis.

$$D = \begin{bmatrix} 0 & 18 & 23 & 25 & 6 \\ 18 & 0 & 15 & 17 & 12 \\ 23 & 15 & 0 & 16 & 17 \\ 25 & 17 & 16 & 0 & 19 \\ 6 & 12 & 17 & 19 & 0 \end{bmatrix}$$

a



b

Phylogenetic Tree Construction from a Distance Matrix, Figure 1
a An additive distance matrix *D* of order 5. b A tree $\mathcal{T}$ which realizes *D*. Here, {1, 2, . . . , 5} forms a subset of the nodes of $\mathcal{T}$

tion algorithms have been developed. A comprehensive description of some of the most popular distance-based methods for phylogenetic reconstruction as well as more background information can be found in, e. g., Chapt. 11 of [6] or Chapt. 4 of [14]. See also [1] and [16] and the references therein.

Cross References

- ► Distance-Based Phylogeny Reconstruction (Fast-Converging)
- ► Distance-Based Phylogeny Reconstruction (Optimal Radius)

Acknowledgments

Supported in part by Kyushu University, JSPS (Japan Society for the Promotion of Science), and INRIA Lille – Nord Europe.

Recommended Reading

1. Abdi, H.: Additive-tree representations. In: Dress, A., von Haeseler, A. (eds.) Trees and Hierarchical Structures: Proceedings of a conference held at Bielefeld, FRG, Oct. 5–9th, 1987. Lecture Notes in Biomathematics, vol. 84, pp. 43–59. Springer (1990)
2. Batagelj, V., Pisanski, T., Simões-Pereira, J.M.S.: An algorithm for tree-realizability of distance matrices. Int. J. Comput. Math. **34**, 171–176 (1990)
3. Bennett, C.H., Li, M., Ma, B.: Chain letters and evolutionary histories. Sci. Am. **288**, 76–81 (2003)
4. Boesch, F.T.: Properties of the distance matrix of a tree. Quarterly Appl. Math. **26**, 607–609 (1968)
5. Culberson, J.C., Rudnicki, P.: A fast algorithm for constructing trees from distance matrices. Inf. Process. Lett. **30**, 215–220 (1989)
6. Felsenstein, J.: Inferring Phylogenies. Sinauer Associates, Inc. (2004)
7. Gusfield, D.M.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press, New York (1997)
8. Hakimi, S.L., Yau, S.S.: Distance matrix of a graph and its realizability. Quarterly Appl. Math. **22**, 305–317 (1964)
9. Hein, J.: An optimal algorithm to reconstruct trees from additive distance data. Bull. Math. Biol. **51**, 597–603 (1989)
10. King, V., Zhang, L., Zhou, Y.: On the complexity of distance-based evolutionary tree construction. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2003), pp. 444–453 (2003)
11. Nakhleh, L., Warnow, T., Ringe, D., Evans, S.N.: A comparison of phylogenetic reconstruction methods on an Indo-European dataset. Trans. Philol. Soc. **103**, 171–192 (2005)
12. Reyzin, L., Srivastava, N.: On the longest path algorithm for reconstructing trees from distance matrices. Inf. Process. Lett. **101**, 98–100 (2007)
13. The Canterbury Tales Project: University of Birmingham, Brigham Young University, University of Münster, New York University, Virginia Tech, and Keio University. Website: http://www.canterburytalesproject.org/
14. Warnow, T.: Some combinatorial optimization problems in phylogenetics. In: Lovász, L., Gyárfás, G., Katona, G., Recski, A., Székely, L. (eds.) Graph Theory and Combinatorial Biology. Bolyai Society Mathematical Studies, vol. 7, pp. 363–413. Bolyai János Matematikai Társulat (1999)
15. Waterman, M.S., Smith, T.F., Singh, M., Beyer, W.A.: Additive evolutionary trees. J. Theor. Biol. **64**, 199–213 (1977)
16. Wu, B.Y., K.-Chao, M., Tang, C.Y.: Approximation and exact algorithms for constructing minimum ultrametric trees from distance matrices. J. Combin. Optim. **3**, 199–211 (1999)

Phylogeny Reconstruction

- ► Distance-Based Phylogeny Reconstruction (Optimal Radius)

Planar Geometric Spanners

2005; Bose, Smid, Gudmundsson

Joachim Gudmundsson[1], Giri Narasimhan[2], Michiel Smid[3]

[1] DMiST, National ICT Australia Ltd, Alexandria, Australia

[2] Department of Computer Science, Florida International University, Miami, FL, USA

[3] School of Computer Science, Carleton University, Ottawa, ON, Canada

Keywords and Synonyms

Geometric network; Dilation; Detour

Problem Definition

Let S be a set of n points in the plane and let G be an undirected graph with vertex set S, in which each edge (u, v) has a weight, which is equal to the Euclidean distance $|uv|$ between the points u and v. For any two points p and q in S, their shortest-path distance in G is denoted by $\delta_G(p, q)$. If $t \geq 1$ is a real number, then G is a *t-spanner* for S if $\delta_G(p, q) \leq t|pq|$ for any two points p and q in S. Thus, if t is close to 1, then the graph G contains close approximations to the $\binom{n}{2}$ Euclidean distances determined by the pairs of points in S. If, additionally, G consists of $O(n)$ edges, then this graph can be considered a sparse approximation to the complete graph on S. The smallest value of t for which G is a t-spanner is called the *stretch factor* (or *dilation*) of G. For a comprehensive overview of geometric spanners, see the book by Narasimhan and Smid [16].

Assume that each edge (u, v) of G is embedded as the straight-line segment between the points u and v. The graph G is said to be *plane* if its edges intersect only at their common vertices.

In this entry, the following two problems are considered:

Problem 1 *Determine the smallest real number $t > 1$ for which the following is true: For every set S of n points in the plane, there exists a plane graph with vertex set S, which is a t-spanner for S. Moreover, design an efficient algorithm that constructs such a plane t-spanner.*

Problem 2 *Determine the smallest positive integer D for which the following is true: There exists a constant t, such that for every set S of n points in the plane, there exists a plane graph with vertex set S and maximum degree at most D, which is a t-spanner for S. Moreover, design an efficient algorithm that constructs such a plane t-spanner.*

Key Results

Let S be a finite set of points in the plane that is in *general position*, i. e., no three points of S are on a line and no four points of S are on a circle. The *Delaunay triangulation* of S is the plane graph with vertex set S, in which (u, v) is an edge if and only if there exists a circle through u and v that does not contain any point of S in its interior. (Since S is in general position, this graph is a triangulation.) The Delaunay triangulation of a set of n points in the plane can be constructed in $O(n \log n)$ time. Dobkin, Friedman and Supowit [10] were the first to show that the stretch factor of the Delaunay triangulation is bounded by a constant: They proved that the Delaunay triangulation is a t-spanner for $t = \pi(1 + \sqrt{5})/2$. The currently best known upper bound on the stretch factor of this graph is due to Keil and Gutwin [12]:

Theorem 1 *Let S be a finite set of points in the plane. The Delaunay triangulation of S is a t-spanner for S, for $t = 4\pi\sqrt{3}/9$.*

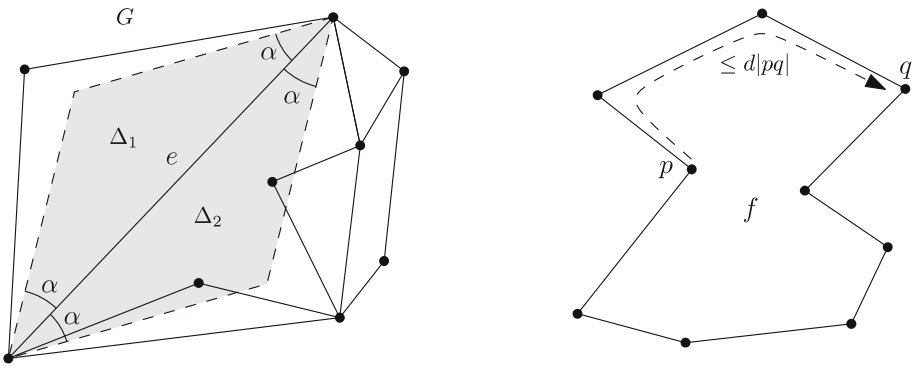
A slightly stronger result was proved by Bose et al. [3]. They proved that for any two points p and q in S, the Delaunay triangulation contains a path between p and q, whose length is at most $(4\pi\sqrt{3}/9)|pq|$ and all edges on this path have length at most $|pq|$.

Levcopoulos and Lingas [14] generalized the result of Theorem 1: Assume that the Delaunay triangulation of the set S is given. Then, for any real number $r > 0$, a plane graph G with vertex set S can be constructed in $O(n)$ time, such that G is a t-spanner for S, where $t = (1+1/r)4\pi\sqrt{3}/9$, and the total length of all edges in G is at most $2r + 1$ times the weight of a minimum spanning tree of S.

The Delaunay triangulation can alternatively be defined to be the dual of the *Voronoi diagram* of the set S. By considering the Voronoi diagram for a metric other than the Euclidean metric, a corresponding Delaunay triangulation is obtained. Chew [7] has shown that the Delaunay triangulation based on the Manhattan-metric is a $\sqrt{10}$-spanner (in this spanner, path-lengths are measured in the Euclidean metric). The currently best result for Problem 1 is due to Chew [8]:

Theorem 2 *Let S be a finite set of points in the plane, and consider the Delaunay triangulation of S that is based on the convex distance function defined by an equilateral triangle. This plane graph is a 2-spanner for S (where path-lengths are measured in the Euclidean metric).*

Das and Joseph [9] have generalized the result of Theorem 1 in the following way (refer to Fig. 1). Let G be a plane graph with vertex set S and let α be a real number with $0 < \alpha < \pi/2$. For any edge e of G, let Δ_1 and Δ_2 be the two isosceles triangles with base e and base angle α. The edge e is said to satisfy the *α-diamond property*, if at least one of the triangles Δ_1 and Δ_2 does not contain any point of S in its interior. The plane graph G is said to satisfy the *α-diamond property*, if every edge e of G satisfies this property. For a real number $d \geq 1$, G satisfies the *d-good polygon property*, if for every face f of G, and for every two vertices p and q on the boundary of f, such that the line segment joining them is completely inside f, the shortest path between p and q along the boundary of f has length at most $d|pq|$. Das and Joseph [9] proved that any plane



Planar Geometric Spanners, Figure 1
On the left, the α-diamond property is illustrated. At least one of the triangles Δ_1 and Δ_2 does not contain any point of S in its interior. On the right, the d-good polygon property is illustrated. p and q are two vertices on the same face f which can see each other. At least one of the two paths between p and q along the boundary of f has length at most $d|pq|$

graph satisfying both the α-diamond property and the d-good polygon property is a t-spanner, for some real number t that depends only on α and d. A slight improvement on the value of t was obtained by Lee [13]:

Theorem 3 *Let $\alpha \in (0, \pi/2)$ and $d \geq 1$ be real numbers, and let G be a plane graph that satisfies the α-diamond property and the d-good polygon property. Then, G is a t-spanner for the vertex set of G, where*

$$t = \frac{8(\pi - \alpha)^2 d}{\alpha^2 \sin^2(\alpha/4)}.$$

To give some examples, it is not difficult to show that the Delaunay triangulation satisfies the α-diamond property with $\alpha = \pi/4$. Drysdale et al. [11] have shown that the minimum weight triangulation satisfies the α-diamond property with $\alpha = \pi/4.6$. Finally, Lee [13] has shown that the greedy triangulation satisfies the α-diamond property with $\alpha = \pi/6$. Of course, any triangulation satisfies the d-good polygon property with $d = 1$.

Now consider Problem 2, that is, the problem of constructing plane spanners whose maximum degree is small. The first result for this problem is due to Bose et al. [2]. They proved that the Delaunay triangulation of any finite point set contains a subgraph of maximum degree at most 27, which is a t-spanner (for some constant t). Li and Wang [15] improved this result, by showing that the Delaunay triangulation contains a t-spanner of maximum degree at most 23. Given the Delaunay triangulation, the subgraphs in [2,15] can be constructed in $O(n)$ time. The currently best result for Problem 2 is by Bose et al. [6]:

Theorem 4 *Let S be a set of n points in the plane. The Delaunay triangulation of S contains a subgraph of maximum degree at most 17, which is a t-spanner for S, for some constant t. Given the Delaunay triangulation of S, this subgraph can be constructed in $O(n)$ time.*

In fact, the result in [6] is more general:

Theorem 5 *Let S be a set of n points in the plane, let $\alpha \in (0, \pi/2)$ be a real number, and let G be a triangulation of S that satisfies the α-diamond property. Then, G contains a subgraph of maximum degree at most $14 + \lceil 2\pi/\alpha \rceil$, which is a t-spanner for S, where t depends only on α. Given the triangulation G, this subgraph can be constructed in $O(n)$ time.*

Applications

Plane spanners have applications in on-line path-finding and routing problems that arise in, for example, geographic information systems and communication networks. In these application areas, the complete environment is not known, and routing has to be done based only on the source, the destination, and the neighborhood of the current position. Bose and Morin [4,5] have shown that, in this model, good routing strategies exist for plane graphs, such as the Delaunay triangulation and graphs that satisfy both the α-diamond property and the d-good polygon property. These strategies are competitive, in the sense that the paths computed have lengths that are within a constant factor of the Euclidean distance between the source and destination. Moreover, these routing strategies use only a limited amount of memory.

Open Problems

None of the results for Problems 1 and 2 that are mentioned in Sect. "Key Results" seem to be optimal. The following problems are open:

1. Determine the smallest real number t, such that the Delaunay triangulation of any finite set of points in the plane is a t-spanner. It is widely believed that $t = \pi/2$. By Theorem 1, $t \leq 4\pi\sqrt{3}/9$.
2. Determine the smallest real number t, such that a plane t-spanner exists for any finite set of points in the plane. By Theorem 2, $t \leq 2$. By taking S to be the set of four vertices of a square, it follows that t must be at least $\sqrt{2}$.
3. Determine the smallest integer D, such that the Delaunay triangulation of any finite set of points in the plane contains a t-spanner (for some constant t) of maximum degree at most D. By Theorem 4, $D \leq 17$. It follows from results in Aronov et al. [1] that the value of D must be at least 3.
4. Determine the smallest integer D, such that a plane t-spanner (for some constant t) of maximum degree at most D exists for any finite set of points in the plane. By Theorem 4 and results in [1], $3 \leq D \leq 17$.

Cross References

- ▶ Applications of Geometric Spanner Networks
- ▶ Dilation of Geometric Networks
- ▶ Geometric Spanners
- ▶ Sparse Graph Spanners

Recommended Reading

1. Aronov, B., de Berg, M., Cheong, O., Gudmundsson, J., Haverkort, H., Vigneron, A.: Sparse geometric graphs with small dilation. In: Proceedings of the 16th International Symposium on Algorithms and Computation. Lecture Notes in Computer Science, vol. 3827, pp. 50–59. Springer, Berlin (2005)
2. Bose, P., Gudmundsson, J., Smid, M.: Constructing plane spanners of bounded degree and low weight. Algorithmica **42**, 249–264 (2005)
3. Bose, P., Maheshwari, A., Narasimhan, G., Smid, M., Zeh, N.: Approximating geometric bottleneck shortest paths. Comput. Geom.: Theory Appl. **29**, 233–249 (2004)
4. Bose, P., Morin, P.: Competitive online routing in geometric graphs. Theor. Comput. Sci. **324**, 273–288 (2004)
5. Bose, P., Morin, P.: Online routing in triangulations. SIAM J. Comput. **33**, 937–951 (2004)
6. Bose, P., Smid, M., Xu, D.: Diamond triangulations contain spanners of bounded degree. In: Proceedings of the 17th International Symposium on Algorithms and Computation. Lecture Notes in Computer Science, vol. 4288, pp. 173–182. Springer, Berlin (2006)
7. Chew, L.P.: There is a planar graph almost as good as the complete graph. In: Proceedings of the 2nd ACM Symposium on Computational Geometry, pp. 169–177 (1986)
8. Chew, L.P.: There are planar graphs almost as good as the complete graph. J. Comput. Syst. Sci. **39**, 205–219 (1989)
9. Das, G., Joseph, D.: Which triangulations approximate the complete graph? In: Proceedings of the International Symposium on Optimal Algorithms. Lecture Notes in Computer Science, vol. 401, pp. 168–192. Springer, Berlin (1989)
10. Dobkin, D.P., Friedman, S.J., Supowit, K.J.: Delaunay graphs are almost as good as complete graphs. Discret. Comput. Geom. **5**, 399–407 (1990)
11. Drysdale, R.L., McElfresh, S., Snoeyink, J.S.: On exclusion regions for optimal triangulations. Discrete Appl. Math. **109**, 49–65 (2001)
12. Keil, J.M., Gutwin, C.A.: Classes of graphs which approximate the complete Euclidean graph. Discrete Comput. Geom. **7**, 13–28 (1992)
13. Lee, A.W.: Diamonds are a plane graph's best friend. Master's thesis, School of Computer Science, Carleton University, Ottawa (2004)
14. Levcopoulos, C., Lingas, A.: There are planar graphs almost as good as the complete graphs and almost as cheap as minimum spanning trees. Algorithmica **8**, 251–256 (1992)
15. Li, X.-Y., Wang, Y.: Efficient construction of low weighted bounded degree planar spanner. Int. J. Comput. Geom. Appl. **14**, 69–84 (2004)
16. Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press, Cambridge, UK (2007)

Planarity Testing

1976; Booth, Lueker

GLENCORA BORRADAILE
Department of Computer Science, Brown University, Providence, RI, USA

Keywords and Synonyms

Planarity testing; Planar embedding

Problem Definition

The problem is to determine whether or not the input graph G is planar. The definition pertinent to planarity-testing algorithms is: G is planar if there is an *embedding* of G into the plane (vertices of G are mapped to distinct points and edges of G are mapped to curves between their respective endpoints) such that edges do not cross. Algorithms that test the planarity of a graph can be modified to obtain such an embedding of the graph.

Key Results

Theorem 1 *There is an algorithm that given a graph G with n vertices, determines whether or not G is planar in $O(n)$ time.*

The first linear-time algorithm was obtained by Hopcroft and Tarjan [5] by analyzing an iterative version of a recursive algorithm suggested by Auslander and Parter [1] and corrected by Goldstein [4]. The algorithm is based on the observation that a connected graph is planar if and only if all its biconnected components are planar. The recursive algorithm works with each biconnected component

in turn: find a separating cycle C and partition the edges of G not in C; define a component of the partition as consisting of edges connected by a path in G that does not use an edge of C; and, recursively consider each cyclic component of the partition. If each component of the partition is planar and the components can be combined with C to give a planar graph, then G is planar.

Another method for determining planarity was suggested by Lempel, Even, and Cederbaum [6]. The algorithm starts with embedding a single vertex and the edges adjacent to this vertex. It then considers a vertex adjacent to one of these edges. For correctness, the vertices must be considered in a particular order. This algorithm was first implemented in $O(n)$ time by Booth and Lueker [2] using an efficient implementation of the PQ-trees data structure. Simpler implementations of this algorithm have been given by Boyer and Myrvold [3] and Shih and Hsu [8].

Tutte gave an algebraic method for giving a *straight-line embedding* of a graph that, if the input graph is 3-connected and planar, is guaranteed to generate a planar embedding. The key idea is to fix the vertices of one face of the graph to be the corners of a convex polygon and then embed every other vertex as the geometric average of its neighbors.

Applications

Planarity testing has applications to computer-aided circuit design and VLSI layout by determining whether a given network can be realized in the plane.

URL to Code

LEDA has an efficient implementation of the Hopcroft and Tarjan planarity testing algorithm [7]: http://www.algorithmic-solutions.info/leda_guide/graph_algorithms/planar_kuratowski.html

Cross References

► Fully Dynamic Planarity Testing

Recommended Reading

1. Auslander, L., Parter, S.V.: On imbedding graphs in the plane. J. Math. and Mech. **10**, pp. 517–523 (1961)
2. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using *PQ*-tree algorithms. J. Comp. Syst. Sci. **13**, pp. 335–379 (1976)
3. Boyer, J., Myrvold, W.: Stop minding your P's and Q's: A simplified O(n) planar embedding algorithm. In: SODA '99: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms. Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 140–146 (1999)
4. Goldstein, A.J.: An efficient and constructive algorithm for testing whether a graph can be embedded in the plane. In: Graph and Combinatorics Conf. (1963)
5. Hopcroft, J., Tarjan, R.: Efficient planarity testing. J. ACM **21**, pp. 549–568 (1974)
6. Lempel, A., Even, S., Cederbaum, I.: An algorithm for planarity testing of graphs. In: Rosentiehl, P. (ed.) Theory of Graphs: International Symposium. New York, Gordon and Breach, pp. 215–232 (1967)
7. Mehlhorn, K., Mutzel, P., Näher, S.: An implementation of the hopcroft and tarjan planarity test. Tech. Rep. MPI-I-93-151, Saarbrücken (1993)
8. Shih, W.-K., Hsu, W.-L.: A new planarity test. Theor. Comput. Sci. **223**, pp. 179–191 (1999)

Point Pattern Matching

2003; Ukkonen, Lemström, Mäkinen

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

Keywords and Synonyms

Point set matching; Geometric matching; Geometric alignment; Largest common point set

Problem Definition

Let $\mathbb{R}$ denote the set of reals and $\mathbb{R}^d$ the d-dimensional real space. A finite subset of $\mathbb{R}^d$ is called a *point set*. The set of all point sets (subsets of $\mathbb{R}^d$) is denoted $\mathcal{P}(\mathbb{R}^d)$.

Point pattern matching problems ask for finding similarities between point sets under some transformations. In the basic set–up a *target* point set $T \subset \mathbb{R}^d$ and a *pattern* point set (*point pattern*) $P \subset \mathbb{R}^d$ are given, and the problem is to locate a subset I of T (if it exists) such that P *matches* I. Matching here means that P becomes exactly or approximately equal to I when a *transformation* from a given set $\mathcal{F}$ of transformations is applied on P.

Set $\mathcal{F}$ can be, for example, the set of all *translations* (a constant vector added to each point in P), or all compositions of translations and *rotations* (after a translation, each point is rotated with respect to a common origin; this preserves the distances and is also called a *rigid movement*), or all compositions of translations, rotations, and *scales* (after translating and rotating, distances to the common origin are multiplied by a constant).

The problem variant with exact matching, called the *Exact Point Pattern Matching (EPPM)* problem, requires that $f(P) = I$ for some $f \in \mathcal{F}$. In other words, the EPPM problem is to decide whether or not there is an allowed transformation f such that $f(P) \subset T$. For example, if $\mathcal{F}$

is the set of translations, the problem is simply to decide whether $P + t \subset T$ for some $t \in \mathbb{R}^d$.

Approximate matching is a better model of many situations that arise in practice. Then the quality of the matching between $f(P)$ and I is controlled using a *threshold parameter* $\varepsilon \geq 0$ and a *distance function* $\delta\colon (\mathcal{P}(\mathbb{R}^d), \mathcal{P}(\mathbb{R}^d)) \to \mathbb{R}$ for measuring distances between point sets. Given $\varepsilon \geq 0$, the *Approximate Point Pattern Matching (APPM)* problem is to determine whether there is a subset $I \subset T$ and a transformation $f \in \mathcal{F}$ such that $\delta(f(P), I) \leq \varepsilon$.

The choice of the distance function δ is another source of diversity in the problem statement. A variant requires that there is a *one-to-one* mapping between $f(P)$ and I, and each point p of $f(P)$ is *ε-close* to its one-to-one counterpart p^* in I, that is, $|p - p^*| \leq \varepsilon$. A commonly studied relaxed version uses matching under a *many-to-one* mapping: it is only required that each point of $f(P)$ has *some* point of I that is *ε-close*; this distance is also known as the *directed Hausdorff distance*. Still more variants come from the choice of the *norm* $|\cdot|$ to measure the distance between points.

Another form of approximation is obtained by allowing a minimum amount of unmatched points in P: The *Largest Common Point Set (LCP)* problem asks for the largest $I \subset T$ such that $I \subset f(P)$ for some $f \in \mathcal{F}$. In the *Largest Approximately Common Point Set (LACP)* problem each point $p^* \in I$ must occur *ε-close* to a point $p \in f(P)$.

Finally, a problem closely related to point pattern matching is to evaluate for point sets A and B their smallest distance $\min_{f \in \mathcal{F}} \delta(f(A), B)$ under transformations $\mathcal{F}$ or to test if this distance is $\leq \varepsilon$. This problem is called the *distance evaluation problem*.

Key Results

A folk theorem is a voting algorithm to solve EPPM under translations in $O(|P||T| \log(|T||P|))$ time: Collect all translations mapping each point of P to each point of T, sort the set, and report the translation getting most votes. If some translation gets $|P|$ votes, then a subset I such $f(P) = I$ is found. With some care in organizing the sorting, one can achieve $O(|P||T| \log |P|)$ time [13].

The voting algorithm also solves the LCP problem under translations. A faster algorithm specific to EPPM is as follows: Let $p_1, p_2, \cdots p_m$ and $t_1, t_2, \cdots t_n$ be the lists of pattern and target points, respectively, *lexicographicly ordered* according to their d-dimensional coordinate values. Consider the translation $f_{i_1} = t_{i_1} - p_1$, for any $1 \leq i_1 \leq n$. One can scan the target points in the lexicographic order to find a point t_{i_2} such that $p_2 + f_{i_1} = t_{i_2}$. If such is found, one can continue scanning from t_{i_2+1} on to find t_{i_3} such that $p_3 + f_{i_1} = t_{i_3}$. This process is continued until a translated point of P does not occur in T or until a translated occurrence of the entire P is found. Careful implementation of this idea leads to the following result showing that the time bound of the naive string matching algorithm is possible also for the exact point pattern matching under translations.

Theorem 1 (Ukkonen et al. 2003 [13]) *The EPPM problem under translations for point pattern P and target T can be solved in $O(mn)$ time and $O(n)$ space where $m = |P| \leq |T| = n$.*

Quadratic running times are probably the best one can achieve for PPM algorithms:

Theorem 2 (Clifford et al. 2006 [10]) *The LCP problem under translations is 3SUM-hard.*

This means that an $o(|P||T|)$ time algorithm for LCP would yield an $o(n^2)$ algorithm for the 3SUM problem, where $|T| = n$ and $|P| = \Theta(n)$. The 3SUM problem asks, given n numbers, whether there are three numbers a, b, and c among them such that $a + b + c = 0$; finding a subquadratic algorithm for 3SUM would be a surprise [5]. For a more in-depth combinatorial characterization of the geometric properties of the EPPM problem, see [7].

For the distance evaluation problems there are plethora of results. An excellent survey of the key results until 1999 is by Alt and Guibas [2]. As an example, consider in the 2-dimensional case how one can decide in $O(n \log n)$ time whether there is a transformation f composed of translation, rotation and scale, such that $f(A) = B$, where $A, B \subset \mathbb{R}^2$ and $n = |A| = |B|$: The idea is to convert A and B into an invariant form such that one can easily check their congruence under the transformations. First, scale is taken into account by scaling A to have the same *diameter* as B (in $O(n \log n)$ time). If A and B are congruent, then they must have the same *centroids* (which can be computed $O(n)$ time). Consider rotating a line from the centroid and listing the angles and distances to other points in the order they are met during the rotation. Having done this (in $O(n \log n)$ time) on both A and B, the lists of angles and distances should be *cyclic shifts* of each other; the list L_A of A occurs as a substring in $L_B L_B$, where L_B is the list of B. This latter step can be done in $O(n)$ time using any linear time exact string matching algorithm. One obtains the following result.

Theorem 3 (Atkinson 1987 [4]) *It is possible to decide in $O(n \log n)$ time whether there is a transformation f com-*

posed of translation, rotation and scale, such that $f(A) = B$, *where* $A, B \subset \mathbb{R}^2$ *and* $|A| = |B| = n$.

Approximate variant of the above problem is much harder. Denote by $f(A) =^{\varepsilon} B$ the *directed approximate congruence* of point sets A and B, meaning that there is a one-to-one mapping from $f(A)$ to B such that for each point in $f(A)$ its image in B is ε-close. The following result demonstrates the added difficulty.

Theorem 4 (Alt et al. 1988 [3]) *It is possible to decide in* $O(n^6)$ *time whether there is a translation* f *such that* $f(A) =^{\varepsilon} B$, *where* $A, B \subset \mathbb{R}^2$ *and* $|A| = |B| = n$. *The same algorithm solves the corresponding LACP problem for point pattern* P *and target* T *under the one-to-one matching condition in* $O((mn)^3)$ *time, where* $m = |P| \leq |T| = n$.

To get an idea of the techniques to achieve the $O((mn)^3)$ time algorithm for LACP, consider first the one-dimensional version, i. e. let $P, T \subset \mathbb{R}$. Observe, that if there is a translation f' such that $f'(P) =^{\varepsilon} T$, then there is a translation f such that $f(P) =^{\varepsilon} T$ *and* a point $p \in P$ that is mapped *exactly* at ε-distance of a point $t \in T$. This lets one concentrate on these $2mn$ *representative translations*. Consider these translations sorted from left to right. Denote the left-most translation by f. Create a bipartite graph, whose nodes are the points in P and in T on the different parties. There is an edge between $p \in P$ and $t \in T$ if and only if $f(p)$ is ε-close to t. Finding a maximum matching in this graph tells the size of the largest approximately common point set after applying the translation f. One can repeat this on each representative translation to find the overall largest common point set. When the representative translations are considered from left to right, the bipartite graph instances are such that one can compute the maximum matchings greedily at each translation in time $O(|P|)$ [6]. Hence, the algorithm solves the one-dimensional LACP problem under translations and one-to-one matching condition in time $O(m^2 n)$, where $m = |P| \leq |T| = n$.

In the two-dimensional case, the set of representative translations is more implicitly defined: In short, the mapping of each point $p \in P$ ε-close to each point $t \in T$, gives mn circles. The boundary of each such circle is partitioned into intervals such that the end points of these intervals can be chosen as representative translations. There are $O((mn)^2)$ such representative translations. As in the one-dimensional case, each representative translation defines a bipartite graph. Once the representative translations along a circle are processed e. g. counterclockwise, the bipartite graph changes only by one edge at a time. This allows an $O(mn)$ time update for the maximum matching at each representative translation yielding an overall $O((mn)^3)$ time algorithm [3].

More efficient algorithms for variants of this problem have been developed by Efrat, Itai, and Katz [11], as by-products of more efficient bipartite matching algorithms for points on a plane. Their main result is the following:

Theorem 5 (Efrat et al. 2001 [11]) *It is possible to decide in* $O(n^5 \log n)$ *time whether there is a translation* f *such that* $f(A) =^{\varepsilon} B$, *where* $A, B \subset \mathbb{R}^2$ *and* $|A| = |B| = n$.

The problem becomes somewhat easier when the one-to-one matching condition is relaxed; one-to-one condition seems to necessitate the use of bipartite matching in one form or another. Without the condition, one can match the points independently of each other. This gives many tools to preprocess and manipulate the point sets during the algorithm using dynamic geometric data structures. Such techniques are exploited e. g. in the following result.

Theorem 6 (Chew and Kedem 1992 [8]) *The LACP problem under translations and using directed Hausdorff distance and the* L_1 *norm, can be solved in* $O(mn \log n)$ *time, where* $P, T \subset \mathbb{R}^2$ *and* $m = |P| \leq |T| = n$. *The distance evaluation problem for directed Hausdorff distance can be solved in* $O(n^2 \log^2 n)$ *time.*

Most algorithms revisited here have relatively high running times. To obtain faster algorithms, it seems that randomization and approximation techniques are necessary. See [9] for a comprehensive summary of the main achievements in that line of development.

Finally, note that the linear transformations considered here are not always enough to model a real-world problem – even when approximate congruence is allowed. Sometimes the proper transformation between two point sets (or between their subsets) is non-linear, without an easily parametrizable representation. Unfortunately, the formulations trying to capture such non-uniformness have been proven NP-hard [1] or even NP-hard to approximate within any constant factor [12].

Applications

Point pattern matching is a fundamental problem that naturally arises in many application domains such as computer vision, pattern recognition, image retrieval, music information retrieval, bioinformatics, dendrochronology, and many others.

Cross References

- ► Assignment Problem
- ► Multidimensional String Matching

► Sequential Exact String Matching
► Stable Matching

Recommended Reading

1. Akutsu, T., Kanaya, K., Ohyama, A., Fujiyama, A.: Point matching under non-uniform distortions. Discret. Appl. Math. **127**, 5–21 (2003)
2. Alt, H., Guibas, L.: Discrete geometric shapes: Matching, interpolation, and approximation. In: Sack, J.R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 121–153. Elsevier Science Publishers B.V. North-Holland, Amsterdam (1999)
3. Alt, H., Mehlhorn, K., Wagener, H., Welzl, E.: Congruence, similarity and symmetries of geometric objects. Discret. Comput. Geom. **3**, 237–256 (1988)
4. Atkinson, M.D.: An optimal algorithm for geometric congruence. J. Algorithms **8**, 159–172 (1997)
5. Barequet, G., Har-Peled, S.: Polygon containment and translational min-hausdorff-distance between segment sets are 3SUM-hard. Int. J. Comput. Geom. Appl. **11**(4), 465–474 (2001)
6. Böcker, S., Mäkinen, V.: Maximum line-pair stabbing problem and its variations. In: Proc. 21st European Workshop on Computational Geometry (EWCG'05), pp. 183–186. Technische Universität Eindhoven, The Netherlands (2005)
7. Brass, P., Pach, J.: Problems and results on geometric patterns. In: Avis, D. et al. (eds.) Graph Theory and Combinatorial Optimization, pp. 17–36. Springer Science + Business Media Inc., NY, USA (2005)
8. Chew, L.P., Kedem, K.: Improvements on geometric pattern matching problems. In: Proc. Scandinavian Workshop Algorithm Theory (SWAT). LNCS, vol. 621, pp. 318–325. Springer, Berlin (1992)
9. Choi, V., Goyal, N.: An efficient approximation algorithm for point pattern matching under noise. In: Proc. 7th Latin American Symposium on Theoretical Informatics (LATIN 2006). LNCS, vol. 3882, pp. 298–310. Springer, Berlin (2006)
10. Clifford, R., Christodoukalis, M., Crawford, T., Meredith, D., Wiggins, G.: A Fast, Randomised, Maximum Subset Matching Algorithm for Document-Level Music Retrieval. In: Proc. International Conference on Music Information Retrieval (ISMIR 2006), University of Victoria, Canada (2006)
11. Efrat, A., Itai, A., Katz, M.: Geometry Helps in Bottleneck Matching and Related Problems. Algorithmica **31**(1), 1–28 (2001)
12. Mäkinen, V., Ukkonen, E.: Local Similarity Based Point-Pattern Matching. In: Proc. 13th Annual Symposium on Combinatorial Pattern Matching (CPM 2002). LNCS, vol. 2373, pp. 115–132. Springer, Berlin (2002)
13. Ukkonen, E., Lemström, K., Mäkinen, V.: Sweepline the music! In: Klein, R. Six, H.W., Wegner, L. (eds.) Computer Science in Perspective, Essays Dedicated to Thomas Ottmann. LNCS, vol. 2598, pp. 330–342. Springer (2003)

Position Auction

2005; Varian

ARIES WEI SUN
Department of Computer Science, City University of Hong Kong, Hong Kong, China

Keywords and Synonyms

Adword auction

Problem Definition

This problem is concerned with the Nash equilibria of a game based on the ad auction used by Google and Yahoo. This research work [5] is motivated by the huge revenue that the adword auction derives every year. It defines two types of Nash equilibrium in the position auction game, applies economic analysis to the equilibria, and provides some empirical evidence that the Nash equilibria of the position auction describes the basic properties of the prices observed in Google's adword auction reasonably accurately. The problem being studied is closely related to the assignment game studied by [4,1,3]. And [2] has independently examined the problem and developed related results.

The Model and its Notations

Consider the problem of assigning agents $a = 1, 2, \ldots, A$ to slots $s = 1, 2, \ldots, S$ where agent a's valuation for slot s is given by $u_{as} = v_a x_s$. The slots are numbered such that $x_1 > x_2 > \ldots > x_S$. It is assumed that $x_S = 0$ for all $s > S$ and the number of agents is greater than the number of slots. A higher position receives more clicks, so x_s can be interpreted as the click-through rate for slot s. The value $v_a > 0$ can be interpreted as the expected profit per click so $u_{as} = v_a x_s$ indicates the expected profit to advertiser a from appearing in slot s.

The slots are sold via an auction. Each agent bids an amount b_a, with the slot with the best click through rate being assigned to the agent with the highest bid, the second-best slot to the agent with the second highest bid, and so on. Renumbering the agents if necessary, let v_s be the value per click of the agent assigned to slot s. The price agent s faces is the bid of the agent immediately below him, so $p_t = b_{t+1}$. Hence the net profit that agent a can expect to make if he acquires slot s is $\left(v_a - p_s\right) x_s = \left(v_a - b_{s+1}\right) x_s$.

Definitions

Definition 1 A Nash equilibrium set of prices ***(NE)*** satisfies

$$\begin{aligned} \left(v_s - p_s\right) x_s &\geq \left(v_s - p_t\right) x_t, \text{ for } t > s \\ \left(v_s - p_s\right) x_s &\geq \left(v_s - p_{t-1}\right) x_t, \text{ for } t < s \end{aligned}$$

where $p_t = b_{t+1}$.

Definition 2 A symmetric Nash equilibrium set of prices ***(SNE)*** satisfies

$$(v_s - p_s)\, x_s \geq (v_s - p_t)\, x_t \text{ for all } t \text{ and } s.$$

Equivalently,

$$v_s\,(x_s - x_t) \geq p_s x_s - p_t x_t \text{ for all } t \text{ and } s.$$

Key Results

Facts of NE and SNE

Fact 1 (Non-negative surplus) In an SNE, $v_s \geq p_s$.
Fact 2 (Monotone values) In an SNE, $v_{s-1} \geq v_s$, for all s.
Fact 3 (Monotone prices) In an SNE, $p_{s-1}x_{s-1} > p_s x_s$ and $p_{s-1} \geq p_s$ for all s. If $v_s > p_s$ then $p_{s-1} > p_s$.
Fact 4 ($NE \supset SNE$) If a set of prices is an SNE then it is an NE.
Fact 5 (One-step solution) If a set of bids satisfies the symmetric Nash equilibria inequalities for $s+1$ and $s-1$, then it satisfies these inequalities for all s.
Fact 6 The maximum revenue NE yields the same revenue as the upper recursive solution to the SNE.

A Sufficient and Necessary Condition of the Existence of a Pure Strategy Nash Equilibrium in the Position Auction Game

Theorem 1 *In the position auction described before, a pure strategy Nash equilibrium exists if and only if all the intervals*

$$\left[\frac{p_s x_s - p_{s+1}x_{s+1}}{x_s - x_{s+1}}, \frac{p_{s-1}x_{s-1} - p_s x_s}{x_{s-1} - x_s}\right], \text{ for } s = 2, 3, \ldots, S$$

are non-empty.

Applications

The model studied in this paper is a simple and elegant abstraction of the real adword auctions used by search engines such as Google and Yahoo. Different search engines have slightly different rules. For example, Yahoo ranks the advertisers according to their bids, while Google ranks the advertisers not only according to their bids but also according to the likelihood of their links being clicked.

However, similar analysis can be applied to real world situations, as the author has demonstrated above for the Google adword auction case.

Cross References

► Adwords Pricing

Recommended Reading

1. Demange, G., Gale, D., Sotomayor, M.: Multi-item auctions. J. Polit. Econ. **94**(4), 863–72 (1986)
2. Edelman, B., Ostrovsky, M., Schwartz, M.: Internet advertising and the generalized second price auction. NBER Working Paper, 11765, November 2005
3. Roth, A., Sotomayor, M.: Two-Sided Matching. Cambridge University Press, Cambridge (1990)
4. Shapely, L., Shubik, M.: The Assignment Game I: the core. Int. J. Game Theor. **1**, 111–130 (1972)
5. Varian, H.R.: Position auctions. Int. J. Ind. Organ. **25**(6), 1163–1178 (2007)

Predecessor Search

2006; Pătraşcu, Thorup

MIHAI PĂTRAŞCU
CSAIL, MIT, Cambridge, MA, USA

Keywords and Synonyms

Predecessor problem Successor problem IP lookup

Problem Definition

Consider an ordered universe U, and a set $T \subset U$ with $|T| = n$. The goal is to preprocess T, such that the following query can be answered efficiently: given $x \in U$, report the predecessor of x, i. e. $\max\{y \in T \mid y < x\}$. One can also consider the dynamic problem, where elements are inserted and deleted into T. Let t_q be the query time, and t_u the update time.

This is a fundamental search problem, with an impressive number of applications. Later, this entry discusses IP lookup (forwarding packets on the Internet), orthogonal range queries and persistent data structures as examples.

The problem was considered in many computational models. In fact, most models below were initially defined to study the predecessor problem.

Comparison model: The problem can be solved through binary search in $\Theta(\lg n)$ comparisons. There is a lot of work on adaptive bounds, which may be sublogarithmic. Such bounds may depend on the finger distance, the working set, entropy etc.
Binary search trees: Predecessor search is one of the fundamental motivations for binary search trees. In this restrictive model, one can hope for an instance optimal (competitive) algorithm. Attempts to achieve this are described in a separate entry.[1]

[1] $O(\log\log n)$-*competitive Binary Search Trees* (2004; Demaine, Harmon, Iacono, Pătraşcu)

Word RAM: Memory is organized as words of b bits, and can be accessed through indirection. Constant-time operations include the standard operations in a language such as C (addition, multiplication, shifts and bitwise operations).

It is standard to assume the universe is $U = \{1, \ldots, 2^\ell\}$, i. e. one deals with ℓ-bit integers. The floating point representation was designed so that order is preserved when values are interpreted as integers, so any algorithm will also work for ℓ-bit floating point numbers.

The standard *transdichotomous* assumption is that $b = \ell$, so that an input integer is represented in a word. This implies $b \geq \lg n$.

Cell-probe model: This is a nonuniform model stronger than the word RAM, in which the operations are arbitrary functions on the memory words (cells) which have already been probed. Thus, t_q only counts the number of cell probes. This is an ideal model for lower bounds, since it does not depend on the operations implemented by a particular computer.

Communication games: Let Alice have the query x, and Bob have the set T. They are trying to find the predecessor of x through τ rounds of communication, where in each round Alice sends m_A bits, and Bob replies with m_B bits.

This can simulate the cell-probe model when $m_B = b$ and m_A is the logarithm of the memory size. Then $\tau \leq t_q$ and one can use communication complexity to obtain cell-probe lower bounds.

External memory: The unit of access is a page, containing B words of ℓ bits each. B-trees solve the problem with query and update time $O(\log_B n)$, and one can also achieve this oblivious to the value of B.[2] The cell-probe model with $b = B \cdot \ell$ is stronger than this model.

AC^0 RAM: This is a variant of the word RAM in which allowable operations are functions that have constant depth, unbounded fan-in circuits. This excludes multiplication from the standard set of operations.

RAMBO: this is a variant of the RAM with a nonstandard memory, where words of memory can overlap in their bits. In the static case this is essentially equivalent to a normal RAM. However, in the dynamic case updates can be faster due to the word overlap [5].

The worst-case logarithmic bound for comparison search is not particularly informative when efficiency really matters. In practice, B-trees and variants are standard when dealing with huge data sets. Solutions based on RAM tricks are essential when the data set is not too large, but a fast query time is crucial, such as in software solutions to IP lookup [7].

[2] See *Cache-oblivious B-tree* (2005; Bender, Demaine, Farach-Colton).

Key Results

Building on a long line of research, Pătraşcu and Thorup [15,16] finally obtained matching upper and lower bounds for the static problem in the word RAM, cell-probe, external memory and communication game models.

Let S be the number of words of space available. (In external memory, this is equivalent to S/B pages.) Define $a = \lg S \cdot \ell / n$. Also define $\lg x = \lceil \log_2(x+2) \rceil$, so that $\lg x \geq 1$ even if $x \in [0,1]$. Then the optimal search time is, up to constant factors:

$$\min \begin{cases} \log_b n = \Theta(\min\{\log_B n, \log_\ell n\}) \\ \lg \frac{\ell - \lg n}{a} \\ \frac{\lg \frac{\ell}{a}}{\lg\left(\frac{a}{\lg n} \cdot \lg \frac{\ell}{a}\right)} \\ \frac{\lg \frac{\ell}{a}}{\lg\left(\frac{\lg \frac{\ell}{a}}{\lg \frac{n}{a}}\right)} \end{cases} \tag{1}$$

The bound is achieved by a deterministic query algorithm. For any space S, the data structure can be constructed in time $O(S)$ by a randomized algorithm, starting with the set T given in sorted order. Updates are supported in expected time $t_q + O(S/n)$. Thus, besides locating the element through one predecessor query, updates change a minimal fraction of the data structure.

Lower bounds hold in the powerful cell-probe model, and hold even for randomized algorithms. When $S \geq n^{1+\varepsilon}$, the optimal trade-off for communication games coincides to (1). Note that the case $S = n^{1+o(1)}$ essentially disappears in the reduction to communication complexity, because Alice's messages only depends on $\lg S$. Thus, there is no asymptotic difference between $S = O(n)$ and, say, $S = O(n^2)$.

Upper Bounds

The following algorithmic techniques give the optimal result:

- *B-trees* give $O(\log_B n)$ query time with linear space.
- *Fusion trees*, by Fredman and Willard [10], achieve a query time of $O(\log_b n)$. The basis of this is a *fusion node*, a structure which can search among b^ε values in constant time. This is done by recognizing that

only a few bits of each value are essential, and packing the relevant information about all values in a single word.

- *Van Emde Boas search* [18] can solve the problem in $O(\lg \ell)$ time by binary searching for the length of the longest common prefix between the query and a value in T. Beginning the search with a table lookup based on the first $\lg n$ bits, and ending when there is enough space to store all answers, the query time is reduced to $O(\lg((\ell - \lg n)/a))$.
- A technique by *Beame and Fich* [4] can perform a multiway search for the longest common prefix, by maintaining a careful balance between ℓ and n. This is relevant when the space is at least $n^{1+\varepsilon}$, and gives the third branch of (1). Pătraşcu and Thorup [15] show how related ideas can be implemented with smaller space, yielding the last branch of (1).

Observe that external memory only features in the optimal trade-off through the $O(\log_B n)$ term coming from B-trees. Thus, it is optimal to either use the standard, comparison-based B-trees, or use the best word RAM strategy which completely ignores external memory.

Lower Bounds

All lower bounds before [15] where shown in the communication game model. Ajtai [1] was the first to prove a superconstant lower bound. His results, with a correction by Miltersen [12], show that for polynomial space, there exists n as a function of ℓ making the query time $\Omega(\sqrt{\lg \ell})$, and likewise there exists ℓ a function of n making the query complexity $\Omega(\sqrt[3]{\lg n})$.

Miltersen et al [13] revisited Ajtai's proof, extending it to randomized algorithms. More importantly, they captured the essence of the proof in an independent *round elimination lemma*, which is an important tool for proving lower bounds in asymmetric communication.

Beame and Fich [4] improved Ajtai's lower bounds to $\Omega(\lg \ell / \lg \lg \ell)$ and $\Omega(\sqrt{\lg n / \lg \lg n})$ respectively. Sen and Venkatesh [17] later gave an improved round elimination lemma, which can reprove these lower bounds, but also for randomized algorithms.

Finally, using the message compression lemma of [6] (an alternative to round elimination), Pătraşcu and Thorup [15] showed an optimal trade-off for communication games. This is also an optimal lower bound in the other models when $S \geq n^{1+\varepsilon}$, but not for smaller space.

More importantly, [15] developed the first tools for proving lower bounds exceeding communication complexity, when $S = n^{1+o(1)}$. This showed the first separation ever between a data structure or polynomial size, and one of near linear size. This is fundamentally impossible through a direct communication lower bound, since the reduction to communication games only depends on $\lg S$.

The full result of Pătraşcu and Thorup [15] it the trade-off (1). Initially, this was shown only for deterministic query algorithms, but eventually it was extended to a randomized lower bound as well [16]. Among the surprising consequences of this result was that the classic van Emde Boas search is optimal for near-linear space (and thus for dynamic data structures), whereas with quadratic space it can be beaten by the technique of Beame and Fich.

A key technical idea of [15] is to analyze many queries simultaneously. Then, one considers a communication game involving all queries, and proves a direct-sum version of the round elimination lemma. Arguably, the proof is even simpler than for the regular round elimination lemma. This is achieved by considering a stronger model for the inductive analysis, in which the algorithm is allowed to *reject* a large fraction of the queries before starting to make probes.

Bucketing

The rich recursive structure of the problem can not only be used for fast queries, but also to optimize the space and update time – of course, within the limits of (1). The idea is to place ranges of consecutive values in buckets, and include a single representative of each bucket in the predecessor structure. After performing a query on the predecessor structure (now with fewer elements), one need only search within the relevant bucket.

Because buckets of size $w^{O(1)}$ can be handled in constant time by fusion trees, it follows that factors of w in space are irrelevant. A more extreme application of the idea is given by *exponential trees* [3]. Here buckets have size $\Theta(n^{1-\gamma})$, where γ is a sufficiently small constant. Buckets are handled recursively in the same way, leading to $O(\lg \lg n)$ levels. If the initial query time is at least $t_q \geq \lg^{\varepsilon} n$, the query times at each level decrease geometrically, so overall time only grows by a constant factor. However, any polynomial space is reduced to linear, for an appropriate choice of γ. Also, the exponential tree can be updated in $O(t_q)$ time, even if the original data structure was static.

Applications

Perhaps the most important application of predecessor search is IP lookup. This is the problem solved by routers for each packet on the Internet, when deciding which subnetwork to forward the packet to. Thus, it is probably the most run algorithmic problem in the world. Formally, this

is an *interval stabbing* query, which is equivalent to predecessor search in the static case [9]. As this is a problem where efficiency really matters, it is important to note that the fastest deployed software solutions [7] use integer search strategies (not comparison-based), as theoretical results would predict.

In addition, predecessor search is used pervasively in data structures, when reducing problems to *rank space*. Given a set T, one often wants to relabel it to the simpler $\{1, \ldots, n\}$ ("rank space"), while maintaining order relations. If one is presented with new values dynamically, the need for a predecessor query arises. Here are a couple of illustrative examples:

- In orthogonal range queries, one maintains a set of points in U^d, and queries for points in some rectangle $[a_1, b_1] \times \cdots \times [a_d, b_d]$. Though bounds typically grow exponentially with the dimension, the dependence on the universe can be factored out. At query time, one first runs $2d$ predecessor queries transforming the universe to $\{1, \ldots, n\}^d$.
- To make pointer data structures persistent [8], an outgoing link is replaced by a vector of pointers, each valid for some period of time. Deciding which link to follow (given the time being queried) is a predecessor problem.

Finally, it is interesting to note that the lower bounds for predecessor hold, by reductions, for all applications described above. To make these reductions possible, the lower bounds are in fact shown for the weaker *colored predecessor* problem. In this problem, the values in T are colored red or blue, and the query only needs to find the color of the predecessor.

Open Problems

It is known [2] how to implement fusion trees with AC^0 instructions, but not the other query strategies. What is the best query trade-off achievable on the AC^0 RAM? In particular, can van Emde Boas search be implemented with AC^0 instructions?

For the dynamic problem, can the update times be made deterministic? In particular, can van Emde Boas search be implemented with fast deterministic updates? This is a very appealing problem, with applications to deterministic dictionaries [14]. Also, can fusion nodes be updated deterministically in constant time? Atomic heaps [11] achieve this when searching only among $(\lg n)^\varepsilon$ elements, not b^ε.

Finally, does an update to the predecessor structure require a query? In other words, can $t_u = o(t_q)$ be obtained, while still maintaining efficient query times?

Cross References

- ► Cache-Oblivious B-Tree
- ► $O(\log \log n)$-competitive Binary Search Tree

Recommended Reading

1. Ajtai, M.: A lower bound for finding predecessors in Yao's cell probe model. Combinatorica **8**(3), 235–247 (1988)
2. Andersson, A., Miltersen, P.B., Thorup, M.: Fusion trees can be implemented with AC^0 instructions only. Theor. Comput. Sci. **215**(1–2), 337–344 (1999)
3. Andersson, A., Thorup, M.: Dynamic ordered sets with exponential search trees. CoRR cs.DS/0210006. See also FOCS'96, STOC'00, 2002
4. Beame, P., Fich, F.E.: Optimal bounds for the predecessor problem and related problems. J. Comput. Syst. Sci. **65**(1), 38–72 (2002). See also STOC'99
5. Brodnik, A., Carlsson, S., Fredman, M.L., Karlsson, J., Munro, J.I.: Worst case constant time priority queue. J. Syst. Softw. **78**(3), 249–256 (2005). See also SODA'01
6. Chakrabarti, A., Regev, O.: An optimal randomised cell probe lower bound for approximate nearest neighbour searching. In: Proc. 45th IEEE Symposium on Foundations of Computer Science (FOCS), 2004, pp. 473–482
7. Degermark, M., Brodnik, A., Carlsson, S., Pink, S.: Small forwarding tables for fast routing lookups. In: Proc. ACM SIGCOMM, 1997, pp. 3–14
8. Driscoll, J.R., Sarnak, N., Sleator, D.D., Tarjan, R.E.: Making data structures persistent. J. Comput. Syst. Sci. **38**(1), 86–124 (1989). See also STOC'86
9. Feldmann, A., Muthukrishnan, S.: Tradeoffs for packet classification. In: Proc. IEEE INFOCOM, 2000, pp. 1193–1202
10. Fredman, M.L., Willard, D.E.: Surpassing the information theoretic bound with fusion trees. J. Comput. Syst. Sci. **47**(3), 424–436 (1993). See also STOC'90
11. Fredman, M.L., Willard, D.E.: Trans-dichotomous algorithms for minimum spanning trees and shortest paths. J. Comput. Syst. Sci. **48**(3), 533–551 (1994). See also FOCS'90
12. Miltersen, P.B.: Lower bounds for Union-Split-Find related problems on random access machines. In: 26th ACM Symposium on Theory of Computing (STOC), 1994, pp. 625–634
13. Miltersen, P.B., Nisan, N., Safra, S., Wigderson, A.: On data structures and asymmetric communication complexity. J. Comput. Syst. Sci. **57**(1), 37–49 (1998). See also STOC'95
14. Pagh, R.: A trade-off for worst-case efficient dictionaries. Nord. J. Comput. **7**, 151–163 (2000). See also SWAT'00
15. Pătraşcu, M., Thorup, M.: Time-space trade-offs for predecessor search. In: Proc. 38th ACM Symposium on Theory of Computing (STOC), 2006, pp. 232–240
16. Pătraşcu, M., Thorup, M.: Randomization does not help searching predecessors. In: Proc. 18th ACM/SIAM Symposium on Discrete Algorithms (SODA), 2007
17. Sen, P., Venkatesh, S.: Lower bounds for predecessor searching in the cell probe model. arXiv:cs.CC/0309033. See also ICALP'01, CCC'03, 2003
18. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and implementation of an efficient priority queue. Math. Syst. Theor. **10**, 99–127 (1977). Announced by van Emde Boas alone at FOCS'75

Price of Anarchy

2005; Koutsoupias

GEORGE CHRISTODOULOU
Max-Planck-Institute for Computer Science,
Saarbruecken, Germany

Keywords and Synonyms

Coordination ratio

Problem Definition

The *Price of Anarchy*, captures the lack of coordination in systems where users are selfish and may have conflicted interests. It was first proposed by Koutsoupias and Papadimitriou in [9], where the term *coordination ratio* was used instead, but later Papadimitriou in [12] coined the term Price of Anarchy, that finally prevailed in the literature.

Roughly, the Price of Anarchy is the system cost (e. g. makespan, average latency) of the worst-case Nash Equilibrium over the optimal system cost, that would be achieved if the players were forced to coordinate. Although it was originally defined in order to analyze a simple load-balancing game, it was soon applied to numerous variants and to more general games. The family of *(weighted) congestion games* [11,13] is a nice abstract form to describe most of the alternative settings.

The Price of Anarchy may vary, depending on the

- equilibirium solution concept *(e. g. pure, mixed, correlated equilibria)*
- characteristics of the congestion game
  - Players Set *(e. g. atomic – non atomic)*
  - Strategy Set *(e. g. symmetric-asymmetric, parallel machines-network-general)*
  - Utility *(e. g. linear, polynomial)*
- social cost *(e. g. maximum, sum, total latency).*

Notations

Let G be a (finite) game, that is determined by the triple $(N, (S_i)_{i\in N}, (c_i)_{i\in N})$. $N = \{1, \ldots, n\}$ is the set of the players, that participate in the game. S_i is a *pure strategy set* for player i. An element $A_i \in S_i$ is a *pure strategy* for player $i \in N$. A *pure strategy profile* $A = (A_1, \ldots, A_n)$ is a vector of pure strategies, one for each player. The set of all possible pure strategy profiles is denoted by $S = S_1 \times \ldots \times S_n$. The *cost* of a player $i \in N$, for a pure strategy, is determined by a cost function $c_i : S \mapsto \mathbb{R}$.

A pure strategy profile A is a *pure Nash equilibrium*, if none of the players $i \in N$ can benefit, by unilaterally deviating to another pure strategy $s_i \in S_i$:

$$c_i(A) \le c_i(A_{-i}, s_i) \qquad \forall i \in N, \quad \forall s_i \in S_i \,,$$

where (A_{-i},s_i) is the simple strategy profile that results when just the player i deviates from strategy $A_i \in S_i$ to strategy $s_i \in S_i$.

A *mixed strategy* p_i for a player $i \in N$, is a probability distribution over her pure strategy set S_i. A mixed strategy profile p is the tuple $p = (p_1, \ldots p_n)$, where player i chooses mixed strategy p_i. The expected cost of a player $i \in N$ with respect to the p, is

$$c_i(p) = \sum_{A\in S} p(A)c_i(A) \,,$$

where $p(A) = \prod_{i\in N} p_i(A_i)$ is the probability that pure strategy A occurs, with respect to $(p_i)_{i\in N}$. A mixed strategy profile p is a *Nash Equilibrium*, if and only if

$$c_i(p) \le c_i(p_{-i}, s_i) \quad \forall i \in N, \ \forall s_i \in S_i \,.$$

The *social cost* of a pure strategy profile A, denoted by $\mathrm{SC}(A)$, is the maximum cost of a player $\textsc{Max}(A) = \max_{i\in N} c_i(A)$ or the average cost of a player. For simplicity, the sum of the players cost is considered (that is n times the average cost) $\textsc{Sum}(A) = \sum_{i\in N} c_i(A)$. The same definitions extend naturally for the case of mixed strategies, but with expected costs in this case.

The (mixed) *Price of Anarchy* [9] for a game, is the worst-case ratio, among all the (mixed) Nash Equilibria, of the social cost over the optimal cost, $\textsc{Opt} = \min_{P\in S} \mathrm{SC}(P)$.

$$\mathrm{PA} = \max_{p \text{ is N.E.}} \frac{\mathrm{SC}(p)}{\textsc{Opt}} \,.$$

The Price of Anarchy for a class of games, is the maximum (supremum) price of anarchy among all the games of this class.

Congestion Games Here, a general class of games is described, that contains most of the games for which Price of Anarchy is studied in the literature. A *congestion game* [11,13], is defined by the tuple $(N, E, (S_i)_{i\in N}, (f_e)_{e\in E})$, where $N = \{1, \ldots, n\}$ is a set of players, E is a set of *facilities*, $S_i \subseteq 2^E$ is the pure strategy set for player i; a pure strategy $A_i \in S_i$ is a subset of the facility set, and f_e is a *cost (or latency) function*[1] with respect to the facility $e \in E$.

[1] Unless otherwise stated, linear cost functions are considered throughout this article. For additional results on more general cost functions see entries ▶ Best Response Algorithms for Selfish Routing, ▶ Computing Pure Equilibria in the Game of Parallel Links, ▶ Price of Anarchy for Routing on Parallel Links

A pure strategy profile $A = (A_1, \ldots, A_n)$ is a vector of pure strategies, one for each player. The cost $c_i(A)$ of player i for the pure strategy profile A is given by

$$c_i(A) = \sum_{e \in A_i} f_e(n_e(A)) ,$$

where $n_e(A)$ is the number of the players that use facility e in A.

In general games, a pure Nash equilibrium may not exist. Rosenthal [13] showed that every congestion game possess at least a pure Nash equilibrium. In particular he defined a potential function over the strategy space

$$\Phi(A) = \sum_{e \in E} \sum_{i=1}^{n_e(A)} f_e(i) .$$

He proved that every local optimum of this potential function is a pure Nash Equilibrium.

$$\Phi(A) \leq \Phi(A_{-i}, s_i), \quad \forall i \in N,\ s_i \in S_i .$$

A congestion game is called *symmetric or single-commodity*, if all the players have the same strategy set: $S_i = C$. The term *asymmetric or multi-commodity* is used, to refer to all the games including the symmetric ones.

A special class of congestion games is the class of *network congestion games*. In this games, the facilities are edges of a (multi)graph $G(V,E)$. The pure strategy set for a player $i \in N$ is the simple paths set from a source $s_i \in V$ to a destination $t_i \in V$. In network symmetric congestion games, all the players have the same source and destination.

A natural generalization of congestion games are the *weighted congestion games*, where every player controls an amount of traffic w_i. The cost of each facility $e \in E$ depends on the total load $\theta_e(A)$ of the facility. In this case, an additional social cost function makes sense, i. e. *total latency*. For a pure strategy profile $A \in S$, the total latency is defined as a weighted sum

$$C(A) = \sum_{e \in E} \theta_e(A) \cdot f_e(\theta_e(A)) .$$

Notice that the sum and the total latency coincide for the case of unweighted congestion games.

In a congestion game with *splittable weights (divisible demands)*, every player $i \in N$, instead of fixing a single pure strategy, she is allowed to distribute her demand among her pure strategy set.

In a *non-atomic congestion game*, there are k different player types $1 \ldots k$. Players are infinitesimal and for each player type i the continuum of the players is denoted by the interval $[0, n_i]$. In general, each player type contributes in a different way to the congestion on the facility $e \in E$, and this contribution is determined by a positive *rate of consumption* $r_{s,e}$ for a strategy $s \in S_i$ and a facility $e \in s$. Each player chooses a strategy that results in a *strategy distribution* $x = (x_s)_{s \in S}$, with $\sum_{s \in S_i} x_s = n_i$.

Key Results

Maximum Social Cost

Here, it is considered the price of anarchy in the case where the social cost is the maximum cost among the players. Formally, for a pure strategy profile A, the social cost is

$$\mathrm{SC}(A) = \textsc{Max}(A) = \max_{i \in N} c_i(A) .$$

The definition naturally extends to mixed strategies.

Theorem 1 ([7,8,9,10]) *The price of anarchy for m identical machines is $\Theta(\log m / \log\log m)$.*

Theorem 2 ([7]) *The price of anarchy for m machines with speeds $s_1 \geq s_2 \geq \ldots \geq s_m$ is*

$$\Theta\left(\min\left\{\frac{\log m}{\log\log\log m}, \frac{\log m}{\log\left(\frac{\log m}{\log(s_1/s_m)}\right)}\right\}\right) .$$

Theorem 3 ([3]) *The price of anarchy for m identical machines, in the asymmetric case is $\Theta(\log m/\log\log m)$ for pure equilibria and $\Theta(\log m/\log\log\log m)$ for mixed equilibria.*

Theorem 4 ([5]) *The price of anarchy for pure equilibria is $\Theta(\sqrt{n})$ for asymmetric but at most 5/2 for symmetric congestion games.*

Theorem 5 ([5]) *The price of anarchy for pure equilibria is at least $\Omega(n^{p/(p+1)})$ and at most $O(n)$ for asymmetric, but at most 5/2 for symmetric congestion games with polynomial latencies.*

Average Social Cost – Total Latency

Here, it is considered as social cost the sum of the players cost (divided by the number of the players)

$$\mathrm{SC}(A) = \textsc{Sum}(A) = \sum_{i \in N} c_i(A)$$

or the weighted sum of the players costs (total latency) for weighted games

$$\mathrm{SC}(A) = \mathrm{C}(A) = \sum_{i \in N} w_i c_i(A) .$$

The definition naturally extends for mixed strategies.

Theorem 6 ([2,4,5]) *The price of anarchy is* 5/2 *for asymmetric and* $(5n-2)/(2n+1)$ *for symmetric congestion games.*

Theorem 7 ([2,4]) *The Price of Anarchy for weighted congestion games is* $1+\phi \approx 2.618$.

Theorem 8 ([6]) *The Price of Anarchy is at most* 3/2 *for congestion games with splittable weights.*

Theorem 9 ([14,15]) *The Price of Anarchy for non-atomic congestion games is* 4/3.

Theorem 10 ([1,2,4]) *The Price of Anarchy for (weighted) congestion games is* $d^{\Theta(p)}$ *for polynomial latencies.*

Applications

The efficiency of large scale networks, in which selfish users interact, is highly affected due to the users' selfish behavior. The Price of Anarchy is a quantitative measure of the lack of coordination in such systems. It is a useful theoretical tool for the analysis and design of telecommunication and traffic networks, where selfish users compete on system's resources motivated by their atomic interests and are indifferent to the social welfare.

Cross References

- ▶ Best Response Algorithms for Selfish Routing
- ▶ Computing Pure Equilibria in the Game of Parallel Links
- ▶ Price of Anarchy for Machines Models

Recommended Reading

1. Aland, S., Dumrauf, D., Gairing, M., Monien, B., Schoppmann, F.: Exact price of anarchy for polynomial congestion games. In: 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS), pp. 218–229. Springer, Marseille (2006)
2. Awerbuch, B., Azar, Y., Epstein A.: Large the price of routing unsplittable flow. In: Proc. of the 37th Annual ACM Symposium on Theory of Computing (STOC), pp. 57–66. ACM, Baltimore (2005)
3. Awerbuch, B., Azar, Y., Richter, Y., Tsur, D.: Tradeoffs in worst-case equilibria. In: Approximation and Online Algorithms, 1st International Workshop (WAOA), pp. 41–52. Springer, Budapest (2003)
4. Christodoulou, G., Koutsoupias, E.: On the price of anarchy and stability of correlated equilibria of linear congestion games. In: Algorithms – ESA 2005, 13th Annual European Symposium, pp. 59–70. Springer, Palma de Mallorca (2005)
5. Christodoulou, G., Koutsoupias, E.: The price of anarchy of finite congestion games. In: Proc. of the 37th Annual ACM Symposium on Theory of Computing (STOC), pp. 67–73. ACM, Baltimore (2005)
6. Cominetti, R., Correa, J.R., Moses, N.E.S.: Network games with atomic players. In: Automata, Languages and Programming, 33rd International Colloquium (ICALP), pp. 525–536. Springer, Venice (2006)
7. Czumaj, A., Vöcking, B.: Tight bounds for worst-case equilibria. In: Proc. of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 413–420. ACM/SIAM, San Fransisco (2002)
8. Koutsoupias, E., Mavronicolas, M., Spirakis, P.G.: Approximate equilibria and ball fusion. Theor. Comput. Syst. **36**, 683–693 (2003)
9. Koutsoupias, E., Papadimitriou, C.H.: Worst-case equilibria. In: Proc. of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS), pp. 404–413. Springer, Trier (1999)
10. Mavronicolas, M., Spirakis, P.G.: The price of selfish routing. In: Proc. on 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 510–519. ACM, Heraklion (2001)
11. Monderer, D., Shapley, L.: Potential games. Games Econ. Behav. **14**, 124–143 (1996)
12. Papadimitriou, C.H.: Algorithms, games, and the internet. In: Proc. on 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 749–753. ACM, Heraklion (2001)
13. Rosenthal, R.W.: A class of games possessing pure-strategy Nash equilibria. Int. J. Game Theor. **2**, 65–67 (1973)
14. Roughgarden, T., Tardos, E.: How bad is selfish routing? J. ACM **49**, 236–259 (2002)
15. Roughgarden, T., Tardos, E.: Bounding the inefficiency of equilibria in nonatomic congestion games. Games Econ. Behav. **47**, 389–403 (2004)

Price of Anarchy for Machines Models

2002; Czumaj, Vöcking

Artur Czumaj[1], Berthold Vöcking[2]

[1] DIMAP and Computer Science, University of Warwick, Coventry, UK

[2] Department of Computer Science, RWTH Aachen University, Aachen, Germany

Keywords and Synonyms

Worst-case coordination ratio; Selfish routing

Problem Definition

Notations

This entry considers a selfish routing model formally introduced by Koutsoupias and Papadimitriou [11], in which the goal is to route the traffic on parallel links with linear latency functions. One can describe this model as a scheduling problem with m independent machines with

speeds $s_1, \ldots, s_m$ and n independent tasks with weights $w_1, \ldots, w_n$. The goal is to allocate the tasks to the machines to minimize the maximum load of the links in the system.

It is assumed that all tasks are assigned by non-cooperative agents. The set of *pure strategies* for task i is the set $\{1, \ldots, m\}$ and a *mixed strategy* is a distribution on this set.

Given a combination $(j_1, \ldots, j_n) \in \{1, \ldots, m\}^n$ of pure strategies, one for each task, the *cost* for task i is $\sum_{j_k=j_i} \frac{w_k}{s_{j_i}}$, which is the time needed for machine j_i chosen by task i to complete all tasks allocated to that machine. Similarly, for a combination of pure strategies $(j_1, \ldots, j_n) \in \{1, \ldots, m\}^n$, the *load* of machine j is defined as $\sum_{j_k=j} \frac{w_k}{s_j}$.

Given n tasks of length $w_1, \ldots, w_n$ and m machines with the speeds $s_1, \ldots, s_m$, let $\mathfrak{opt}$ denote the *social optimum*, that is, the minimum cost over all combinations of pure strategies:

$$\mathfrak{opt} = \min_{(j_1,\ldots,j_n)\in\{1,\ldots,m\}^n} \max_{1\le j\le m} \sum_{i:j_i=j} \frac{w_i}{s_j} .$$

For example, if all machines have the same unit speed ($s_j = 1$ for every j, $1 \le j \le m$) and all tasks have the same unit weight ($w_i = 1$ for every i, $1 \le i \le n$), then the social optimum is $\lceil \frac{n}{m} \rceil$.

It is also easy to see that in any system

$$\mathfrak{opt} \ge \frac{\max_i w_i}{\max_j s_j} .$$

It is known that computing the social optimum is $\mathcal{NP}$-hard even for identical speeds (see [11]).

For mixed strategies, let p_i^j denote the probability that an agent i sends the entire traffic w_i to a machine j. Let ℓ_j denote the *expected load* on a machine j, that is,

$$\ell_j = \frac{1}{s_j} \cdot \sum_{i=1}^{n} w_i p_i^j .$$

For a task i, the *expected cost of task i on machine j* is equal to

$$c_i^j = \frac{w_i}{s_j} + \sum_{t\ne i} \frac{w_t p_t^j}{s_j} = \ell_j + (1 - p_i^j)\frac{w_i}{s_j} .$$

The expected cost c_i^j corresponds to the expected finish time of task i on machine j under the processor sharing scheduling policy. This is an appropriate cost model with respect to the underlying traffic routing application.

Definition 1 (Nash equilibrium) The probabilities $(p_i^j)_{1\le i\le n, 1\le j\le m}$ define a *Nash equilibrium* if and only if any task i will assign non-zero probabilities only to machines that minimize c_i^j, that is, $p_i^j > 0$ implies $c_i^j \le c_i^q$, for every q, $1 \le q \le m$.

As an example, in the system considered above in which all machines have the same unit speed and all weights are the same, the uniform probabilities $p_i^j = \frac{1}{m}$ for all $1 \le j \le m$ and $1 \le i \le n$ define a system in a Nash equilibrium.

The existence of a Nash equilibrium over mixed strategies for non-cooperative games was shown by Nash [13]. In fact, the routing game considered here admits an equilibrium even if all players are restricted to pure strategies, what has been shown by Fotakis et al. [7].

Fix an arbitrary Nash equilibrium, that is, fix the probabilities $(p_i^j)_{1\le i\le n, 1\le j\le m}$ that define a Nash equilibrium. Consider the randomized allocation strategies in which each task i is allocated to a single machine chosen independently at random according to the probabilities p_i^j, that is, task i is allocated to machine j with probability p_i^j. Let C_j, $1 \le j \le m$, be the random variable indicating the *load of machine j* in our random experiment. Observe that C_j is the weighted sum of independent 0–1 random variables J_i^j, $\mathbf{Pr}[J_i^j = 1] = p_i^j$, such that

$$C_j = \frac{1}{s_j} \sum_{i=1}^{n} w_i \cdot J_i^j .$$

Let c denote the *maximum expected load* over all machines, that is,

$$\mathsf{c} = \max_{1\le j\le m} \ell_j .$$

Notice that $\mathbf{E}[C_j] = \ell_j$, and therefore $\mathsf{c} = \max_{1\le j\le m} \mathbf{E}[C_j]$.

Finally, let the *social cost* C be defined as the expected maximum load (instead of maximum expected load), that is,

$$\mathsf{C} = \mathbf{E}[\max_{1\le j\le m} C_j] .$$

Observe that $c \le C$ and possibly $c \ll C$. The goal is to estimate the *price of anarchy* (also called the *worst-case coordination ratio*) which is the worst-case ratio

$$R = \max \frac{\mathsf{C}}{\mathfrak{opt}} ,$$

where the maximum is over all Nash equilibria.

Key Results

Early Work

The study of the price of anarchy has been initiated by Koutsoupias and Papadimitriou [11], who showed also some very basic results for this model. For example, they proved that for two identical machines the price of anarchy is exactly $\frac{3}{2}$, and for two machines (with possibly different speeds) the price of anarchy is at least $\phi = (1+\sqrt{5})/2$. Koutsoupias and Papadimitriou showed also that for m identical machines the price of anarchy is $\Omega(\log m/(\log\log m))$ and it is at most $\mathcal{O}(\sqrt{m \ln m})$, and for m arbitrary machines the price of anarchy is $\mathcal{O}(\sqrt{s_1/s_m \sum_{j=1}^{m} s_j/s_m}\sqrt{\log m})$, where $s_1 \geq s_2 \geq \cdots \geq s_m$ [11].

Koutsoupias and Papadimitriou [11] conjectured also that the price of anarchy for m *identical machines* is $\Theta(\log m/(\log\log m))$. In the quest to resolve this conjecture, Mavronicolas and Spirakis [12] considered the problem in the so-called *fully-mixed model*, which is a special class of Nash equilibria in which all p_i^j are strictly positive. In this model, Mavronicolas and Spirakis [12] showed that for m identical machines in the fully-mixed Nash equilibrium the price of anarchy is $\Theta(\log m(\log\log m))$. Similarly, they proved also that for m (not necessarily identical) machines and n identical weights in the fully-mixed Nash equilibrium, if $m \leq n$, then the price of anarchy is $\Theta(\log n/(\log\log n))$.

The motivation behind studying fully-mixed equilibria is the so-called fully-mixed Nash equilibrium conjecture stating that these equilibria maximize the price of anarchy because they maximize the randomization. The conjecture seems to be quite appealing as a fully-mixed equilibrium can be computed in polynomial time, which led to numerous studies of this kind of equilibria with the hope to obtain efficient algorithms for computing or approximating the price of anarchy with respect to mixed equilibria. However, Fischer and Vöcking [6] disproved the fully-mixed Nash equilibrium conjecture and showed that there is a mixed Nash equilibrium whose expected cost is larger than the expected cost of the fully-mixed Nash equilibrium by a factor of $\Omega(\log m/(\log\log m))$. Furthermore, they presented polynomial time algorithms for approximating the price of anarchy for mixed equilibria on identical machines up to a constant factor.

Tight Bounds for the Price of Anarchy

Czumaj and Vöcking [4] entirely resolved the conjecture of Koutsoupias and Papadimitriou [11] and gave an exact description of the price of anarchy as a function of the number of machines and the ratio of the speed of the fastest machine over the speed of the slowest machine.[1]

Theorem 1 [4] **(Upper Bound)** *The price of anarchy for m machines is bounded from above by*

$$\mathcal{O}\left(\min\left\{\frac{\log m}{\log\log\log m}, \frac{\log m}{\log\left(\frac{\log m}{\log(s_1/s_m)}\right)}\right\}\right),$$

where it is assumed that the speeds satisfy $s_1 \geq \cdots \geq s_m$.

In particular, the price of anarchy for m machines is $\mathcal{O}(\frac{\log m}{\log\log\log m})$.

The theorem follows directly from the following two results [4]: that the maximum expected load c satisfies

$$\begin{aligned} \mathsf{c} &= \mathfrak{opt} \cdot \Gamma^{(-1)}(m) \\ &= \mathfrak{opt} \cdot \mathcal{O}\left(\min\left\{\frac{\log m}{\log\log m}, \log\left(\frac{s_1}{s_m}\right)\right\}\right) \end{aligned}$$

and that the social cost C satisfies

$$\mathsf{C} = \mathfrak{opt} \cdot \mathcal{O}\left(\frac{\log m}{\log\left(\frac{\mathfrak{opt}\cdot\log m}{\mathsf{c}}\right)} + 1\right).$$

If one applied these results to systems in which all agents follow only *pure strategies*, then since then $\ell_j = C_j$ for every j, it holds that $\mathsf{C} = \mathsf{c}$. This leads to the following result.

Corollary 2 [4] *For pure strategies the price of anarchy for m machines is upper bounded by*

$$\mathcal{O}\left(\min\left\{\frac{\log m}{\log\log m}, \log\left(\frac{s_1}{s_m}\right)\right\}\right),$$

where it is assumed that the speeds satisfy $s_1 \geq \cdots \geq s_m$.

Theorem 4 below proves that this corollary gives an asymptotically tight bound for the price of anarchy for pure
strategies.

By Theorem 1, in the special case when all *machines are identical*, the price of anarchy is $\mathcal{O}(\log m/(\log\log m))$;

[1] To simplify the notation, for any real $x \geq 0$, let $\log x$ denote $\log x = \max\{\log_2 x, 1\}$. Also, following standard convention, $\Gamma(N)$ is used to denote the *Gamma (factorial) function*, which for any natural N is defined by $\Gamma(N+1) = N!$ and for an arbitrary real $x > 0$ is $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\, dt$. For the inverse of the Gamma function, $\Gamma^{(-1)}(N)$, it is known that $\Gamma^{(-1)}(N) = x$ such that $\lfloor x \rfloor! \leq N - 1 \leq \lceil x \rceil!$. It is well known that $\Gamma^{(-1)}(N) = (\log N)/(\log\log N)(1 + o(1))$.

this result has been also obtained independently by Koutsoupias et al. [10]. However, in this special case one can get a stronger bound that is tight up to an additive constant.

Theorem 3 [4] *For m identical machines the price of anarchy is at most*

$$\Gamma^{(-1)}(m) + \Theta(1) = \frac{\log m}{\log\log m} \cdot (1 + o(1)) .$$

One can obtain a lower bound for the price of anarchy for *m identical machines* by considering the system in which $p_i^j = \frac{1}{m}$ for every i, j. The Result of Gonnet [9] implies that then the price of anarchy is $\Gamma^{(-1)}(m) - \frac{3}{2} + o(1)$, which implies that Theorem 3 is tight up to an additive constant.

The next theorem shows that the upper bound in Theorem 1 is asymptotically tight.

Theorem 4 [4] **(Lower bound)** *The price of anarchy for m machines is lower bounded by*

$$\Omega\left(\min\left\{\frac{\log m}{\log\log\log m}, \frac{\log m}{\log\left(\frac{\log m}{\log(s_1/s_m)}\right)}\right\}\right) .$$

In particular, the price of anarchy for m machines is $\Omega(\log m/(\log\log\log m))$.

In fact, it can be shown [4] (analogously to the upper bound) that for every positive integer m, positive real r, and $S \geq 1$, there exists a set of m machines with $\frac{s_1}{s_m} = S$ being in a Nash equilibrium and satisfying $\mathfrak{opt} = r$,

$$\mathfrak{c} = \mathfrak{opt} \cdot \Omega\left(\min\left\{\frac{\log m}{\log\log m}, \log\left(\frac{s_1}{s_m}\right)\right\}\right) ,$$

and

$$\mathfrak{C} = \mathfrak{opt} \cdot \Omega\left(\frac{\log m}{\log\left(\frac{\mathfrak{opt}\cdot\log m}{c}\right)}\right) .$$

Applications

The model discussed here has been extended in the literature in numerous ways, in particular in [1,5,8]; see also survey presentations in [3,14].

Open Problems

An interesting attempt that adds an algorithmic or constructive element to the analysis of the price of anarchy is made in [2]. The idea behind "coordination mechanisms" is not to study the price of anarchy for a fixed system, but to design the system in such a way that the increase in cost or the loss in performance due to selfish behavior is as small as possible. This is a promising direction of research that might result in practical guidelines of how to build a distributed system that does not suffer from selfish behavior but might even exploit the selfishness of the agents.

Cross References

- ▶ Computing Pure Equilibria in the Game of Parallel Links
- ▶ Price of Anarchy

Recommended Reading

1. Awerbuch, B., Azar, Y., Richter, Y., Tsur, D.: Tradeoffs in worst-case equlibria. Theor. Comput. Sci. **361**, 200–209 (2006)
2. Christodoulou, G., Koutsoupias, E., Nanavati, A.: Coordination mechanisms. In: Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP), pp. 345–357 (2004)
3. Czumaj, A.: Selfish routing on the Internet. In: Leung, J. (ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Boca Raton, FL, USA (2004)
4. Czumaj, A., Vöcking, B.: Tight bounds for worst-case equilibria. ACM Trans. Algorithms **3**(1) (2007)
5. Czumaj, A., Krysta, P., Vöcking, B.: Selfish traffic allocation for server farms. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), pp. 287–296 (2002)
6. Fischer, S., Vöcking, B.: On the structure and complexity of worst-case equilibria. Theor. Comput. Sci. **378**(2), 165–174 (2007)
7. Fotakis, D., Kontogiannis, S., Koutsoupias, E., Mavronicolas, M., Spirakis, P.: The structure and complexity of Nash equilibria for a selfish routing game. In Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP), pp. 123–134, (2002)
8. Gairing, M., Lücking, T., Mavronicolas, M., Monien, B.: The price of anarchy for polynomial social cost. Theor. Comput. Sci. **369**(1-3), 116–135 (2006)
9. Gonnet, G.: Expected length of the longest probe sequence in hash code searching. J. Assoc. Comput. Mach. **28**(2), 289–304 (1981)
10. Koutsoupias, E., Mavronicolas, M., Spirakis, P.: Approximate equilibria and ball fusion. Theor. Comput. Syst. **36**(6), 683–693 (2003)
11. Koutsoupias, E., Papadimitriou, C.H.: Worst-case equilibria. In: Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS), pp. 404–413 (1999)
12. Mavronicolas, M., Spirakis, P.: The price of selfish routing. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 510–519 (2001)
13. Nash Jr., J.F.: Non-cooperative games. Ann. Math. **54**(2), 286–295 (1951)
14. Vöcking, B.: Selfish load balancing. In: Nisan, N., Roughgarden, T., Tardos, É., Vazirani, V. (eds.) Algorithmic Game Theory. Cambridge University Press, New York, NY, USA (2007)

Probabilistic Data Forwarding in Wireless Sensor Networks

2004; Chatzigiannakis, Dimitriou, Nikoletseas, Spirakis

Sotiris Nikoletseas
Computer Engineering and Informatics, Department and CTI, University of Patras, Patras, Greece

Keywords and Synonyms

Data propagation; Routing

Problem Definition

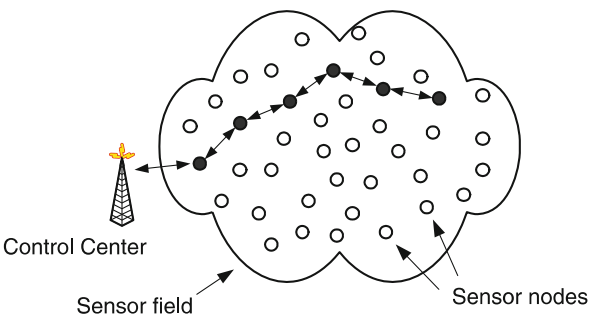
An important problem in wireless sensor networks is that of *local detection and propagation*, i. e. the local sensing of a crucial event and the energy and time efficient propagation of data reporting its realization to a control center (for a graphical presentation, see Fig. 1). This center (called the *"sink"*) could be some human authorities responsible of taking action upon the realization of the crucial event. More formally:

Definition 1 Assume that a single sensor, E, senses the realization of a *local event* $\mathcal{E}$. Then the *propagation problem* is the following: "How can sensor P, via cooperation with the rest of the sensors in the network, efficiently propagate information reporting the realization of the event to the sink S?"

Note that this problem is in fact closely related to the more general problem of data propagation in sensor networks.

Wireless Sensor Networks

Recent dramatic developments in micro-electro-mechanical systems (MEMS), wireless communications and digital electronics have led to the development of small in size, low-power, low-cost sensor devices. Such extremely small (soon in the cubic millimetre scale) devices integrate sensing, data processing and wireless communication capabilities. Examining each such device individually might appear to have small utility, however the effective *distributed self-organization* of large numbers of such devices into an ad-hoc network may lead to the efficient accomplishment of large sensing tasks. Their wide range of applications is based on the use of various sensor types (i. e. thermal, visual, seismic, acoustic, radar, magnetic, etc.) to monitor a wide variety of conditions (e. g. temperature, object presence and movement, humidity, pressure, noise levels etc.). For a survey on wireless sensor networks see [1] and also [6,9].



Probabilistic Data Forwarding in Wireless Sensor Networks, Figure 1
A sensor network

A Simple Model

Sensor networks are comprised of a vast number of ultra-small homogeneous sensors, which are called *"grain" particles*. Each grain particle is a fully-autonomous computing and communication device, characterized mainly by its available power supply (battery) and the energy cost of computation and transmission of data. Such particles (in the model here) do not move. Each particle is equipped with a set of monitors (sensors) for light, pressure, humidity, temperature etc. and has a *broadcast* (digital radio) *beacon mode*.

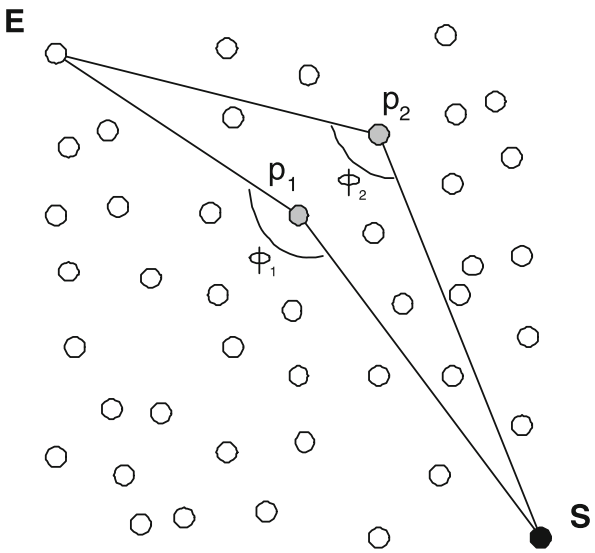
It is assumed that grain particles are *randomly deployed* in a given area of interest. Such a placement may occur e. g. when throwing sensors from an airplane over an area. A special case is considered, when the network being a lattice (or grid) deployment of sensors. This grid placement of grain particles is motivated by certain applications, where it is possible to have a pre-deployed sensor network, where sensors are put (possibly by a human or a robot) in a way that they form a *2-dimensional lattice*.

It is assumed that each particle has the following abilities: (i) It can estimate the direction of a received transmission (e. g. via the technology of direction-sensing antennae). (ii) It can estimate the distance from a nearby particle that did the transmission (e. g. via estimation of the attenuation of the received signal). (iii) It knows the direction towards the sink S. This can be implemented during a set-up phase, where the (powerful) sink broadcasts the information about itself to all particles. (iv) All particles have a common co-ordinates system. Notice that GPS information is not assumed. Also, there is no need to know the global structure of the network.

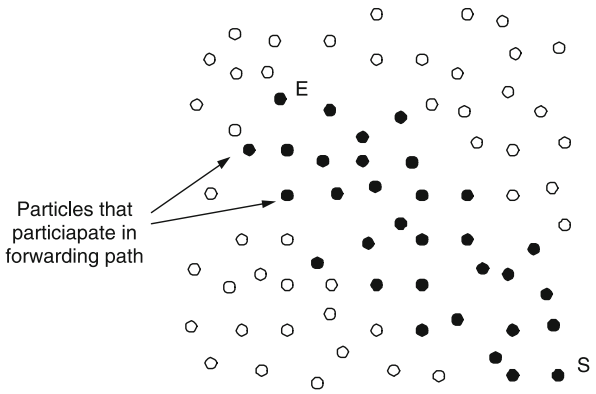
Key Results

The Basic Idea

For the above problem [3] proposes a protocol which tries to minimize energy consumption by *probabilistically fa-*



Probabilistic Data Forwarding in Wireless Sensor Networks, Figure 2
Angle φ and proximity to the optimal line



Probabilistic Data Forwarding in Wireless Sensor Networks, Figure 3
Thin zone of particles

voring certain paths of local data transmissions towards the sink. Thus this protocol is called PFR (Probabilistic Forwarding Protocol). Its basic idea is to *avoid flooding* by favoring (in a probabilistic manner) data propagation along sensors which lie "close" to the (optimal) transmission line, *ES*, that connects the sensor node detecting the event, *E*, and the sink, *S*. This is implemented by locally calculating the angle $\phi = (\widehat{EPS})$, whose corner point *P* is the sensor currently running the local protocol, having received a transmission from a nearby sensor, previously possessing the event information (see Fig. 2). If φ is equal or greater to a predetermined threshold, then *p* will transmit (and thus propagate the information further). Else, it decides whether to transmit with probability equal to $\frac{\phi}{\pi}$. Because of the probabilistic nature of data propagation decisions and to prevent the propagation process from early failing, the protocol initially uses (for a short time period which is evaluated) a flooding mechanism that leads to a sufficiently large *"front"* of sensors possessing the data under propagation. When such a "front" is created, probabilistic Forwarding is performed.

The PFR Protocol

The protocol evolves in two phases:

Phase 1: The "Front" Creation Phase Initially the protocol builds (by using a limited, in terms of rounds, flooding) a sufficiently large "front" of particles, to guarantee the survivability of the data propagation process. During this phase, each particle having received the data to be propagated, deterministically forwards them towards the sink.

Phase 2: The Probabilistic Forwarding Phase Each particle *P* possessing the information under propagation (called $info(\mathcal{E})$ hereafter), calculates an angle φ by calling the subprotocol "φ-calculation" (see description below) and broadcasts $info(\mathcal{E})$ to all its neighbors with probability $\mathbb{P}_{fwd}$ (or it does not propagate any data with probability $1 - \mathbb{P}_{fwd}$) as follows:

$$\mathbb{P}_{fwd} = \begin{cases} 1 & \text{if } \phi \geq \phi_{\text{threshold}} \\ \frac{\phi}{\pi} & \text{otherwise} \end{cases}$$

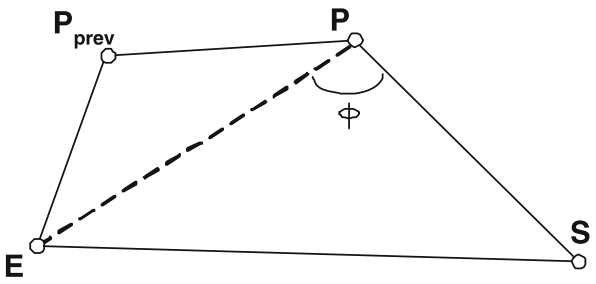
where φ is the $(\widehat{EPS})$ angle and $\phi_{\text{threshold}} = 134°$ (the selection reasons of this value are discussed in [3]).

If the density of particles is appropriately large, then for a line *ES* there is (with high probability) a sequence of points "closely surrounding *ES*" whose angles φ are larger than $\phi_{\text{threshold}}$ and so that successive points are within transmission range. All such points broadcast and thus essentially they follow the line *ES* (see Fig. 3).

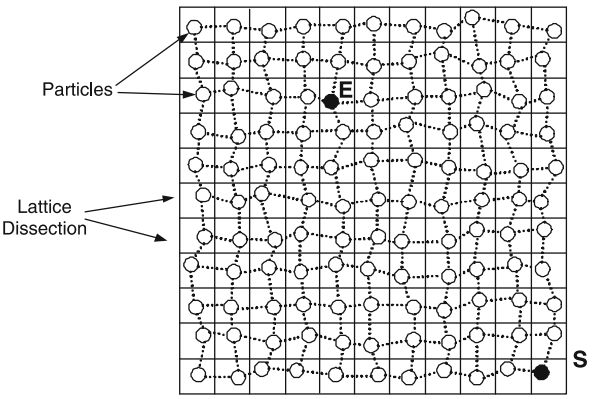
The φ-calculation Subprotocol (see Fig. 4)

Let P_{prev} the particle that transmitted $info(E)$ to *P*.

1. When P_{prev} broadcasts $info(E)$, it also attaches the info $|EP_{\text{prev}}|$ and the direction $\overrightarrow{P_{\text{prev}}E}$.
2. *P* estimates the direction and length of line segment $P_{\text{prev}}P$, as described in the model.
3. *P* now computes angle $(\widehat{EP_{\text{prev}}P})$, and computes $|EP|$ and the direction of $\overrightarrow{PE}$ (this will be used in further transmission from *P*).



Probabilistic Data Forwarding in Wireless Sensor Networks, Figure 4
Angle φ calculation example



Probabilistic Data Forwarding in Wireless Sensor Networks, Figure 5
A lattice dissection *G*

4. P also computes angle $(\widehat{P_{\text{prev}}PE})$ and by subtracting it from $(\widehat{P_{\text{prev}}PS})$ it finds φ.

Performance Properties of PFR

Any protocol Π solving the data propagation problem must satisfy the following three properties: **a) Correctness**. Π must guarantee that data arrives to the position S, given that the whole network exists and is operational. **b) Robustness**. Π must guarantee that data arrives at enough points in a small interval around S, in cases where part of the network has become inoperative. **c) Efficiency**. If Π activates k particles during its operation then Π should have a small ratio of the number of activated over the total number of particles $r = \frac{k}{N}$. Thus r is an energy efficiency measure of Π. It is shown that this is indeed the case for PFR.

Consider a partition of the network area into small squares of a fictitious grid G (see Fig. 5). When particle density is high enough, occupancy arguments guarantee that with very high probability (tending to 1) all squares get particles. All the analysis is conditioned on this event, call it F, of at least one particle in each square. Below only sketches of proofs are provided (full proofs can be found in [3]).

The Correctness of PFR

Consider any square Σ intersecting the ES line. By the occupancy argument above, there is w.h.p. a particle in this square. Clearly, the worst case is when the particle is located in one of the corners of Σ (since the two corners located most far away from the ES line have the smallest φ-angle among all positions in Σ). By geometric calculations, [3] proves that the angle φ of this particle is $\phi > 134°$. But the initial square (i. e. that containing E) always broadcasts and any intermediate intersecting square will be notified (by induction) and thus broadcast because of the argument above. Thus the sink will be reached if the whole network is operational:

Lemma 1 ([3]) *PFR succeeds with probability 1 given the event F.*

The Energy Efficiency of PFR

Consider a "lattice-shaped" network like the one in Fig. 5 (all results will hold for any random deployment "in the limit"). The analysis of the energy efficiency considers particles that are active but are as far as possible from ES. [3] estimates an upper bound on the number of particles in an $n \times n$ (i. e. $N = n \times n$) lattice. If k is this number then $r = \frac{k}{n^2}$ $(0 < r \leq 1)$ is the "energy efficiency ratio" of PFR. More specifically, in [3] the authors prove the (very satisfactory) result below. They consider the area around the ES line, whose particles participate in the propagation process. The number of active particles is thus, roughly speaking, captured by the size of this area, which in turn is equal to $|ES|$ times the maximum distance from $|ES|$. This maximum distance is clearly a random variable. To calculate the expectation and variance of this variable, the authors in [3] basically "upper bound" the stochastic process of the distance from ES by a random walk on the line, and subsequently "upper bound" this random walk by a well-known stochastic process (i. e. the "discouraged arrivals" birth and death Markovian process. Thus they prove:

Theorem 2 ([3]) *The energy efficiency of the PFR protocol is* $\Theta\left(\left(\frac{n_0}{n}\right)^2\right)$ *where* $n_0 = |ES|$ *and* $n = \sqrt{N}$, *where N is the number of particles in the network. For* $n_0 = |ES| = o(n)$, *this is o(1).*

The Robustness of PFR

Consider particles "very near" to the *ES* line. Clearly, such particles have large φ-angles (i. e. $\phi > 134°$). Thus, even in the case that some of these particles are not operating, the probability that none of those operating transmits (during phase 2) is very small. Thus:

Lemma 3 ([3]) *PFR manages to propagate the crucial data across lines parallel to ES, and of constant distance, with* fixed *nonzero probability (not depending on n, |ES|).*

Applications

Sensor networks can be used for continuous sensing, event detection, location sensing as well as micro-sensing. Hence, sensor networks have several important applications, including (a) security (like biological and chemical attack detection), (b) environmental applications (such as fire detection, flood detection, precision agriculture), (c) health applications (like telemonitoring of human physiological data) and (d) home applications (e. g. smart environments and home automation). Also, sensor networks can be combined with other wireless networks (like mobile) or fixed topology infrastructures (like the Internet) to provide transparent wireless extensions in global computing scenaria.

Open Problems

It would be interesting to come up with formal models for sensor networks, especially with respect to energy aspects; in this respect, [10] models energy dissipation using stochastic methods. Also, it is important to investigate fundamental trade-offs, such as those between energy and time. Furthermore, the presence of mobility and/or multiple sinks (highly motivated by applications) creates new challenges (see e. g. [2,11]). Finally, heterogeneity aspects (e. g. having sensors of various types and/or combinations of sensor networks with other types of networks like p2p, mobile and the Internet) are very important; in this respect see e. g. [5,13].

Experimental Results

An implementation of the PFR protocol along with a detailed comparative evaluation (using simulation) with greedy forwarding protocols can be found in [4]; with clustering protocols (like LEACH, [7]) in [12]; with tree maintenance approaches (like Directed Diffusion, [8]) in [5]. Several performance measures are evaluated, like the success rate, the latency and the energy dissipation. The simulations mainly suggest that PFR behaves best in sparse networks of high dynamics.

Cross References

- ▶ Communication in Ad Hoc Mobile Networks Using Random Walks
- ▶ Obstacle Avoidance Algorithms in Wireless Sensor Networks
- ▶ Randomized Energy Balance Algorithms in Sensor Networks

Recommended Reading

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. J. Comput. Netw. **38**, 393–422 (2002)
2. Chatzigiannakis, I., Kinalis, A., Nikoletseas, S.: Sink Mobility Protocols for Data Collection in Wireless Sensor Networks . In: Proc. of the 4th ACM/IEEE International Workshop on Mobility Management and Wireless Access Protocols (MobiWac), ACM Press, pp. 52–59 (2006)
3. Chatzigiannakis, I., Dimitriou, T., Nikoletseas, S., Spirakis, P.: A Probabilistic Algorithm for Efficient and Robust Data Propagation in Smart Dust Networks. In: Proc. 5th European Wireless Conference on Mobile and Wireless Systems (EW 2004), pp. 344–350 (2004). Also in: Ad-Hoc Netw J **4**(5), 621–635 (2006)
4. Chatzigiannakis, I., Dimitriou, T., Mavronicolas, M., Nikoletseas, S., Spirakis, P.: A Comparative Study of Protocols for Efficient Data Propagation in Smart Dust Networks. In: Proc. 9th European Symposium on Parallel Processing (EuroPar), Distinguished Paper. Lecture Notes in Computer Science, vol. 2790, pp. 1003–1016. Springer (2003) Also in the Parallel Processing Letters (PPL) Journal, Volume 13, Number 4, pp. 615–627 (2003)
5. Chatzigiannakis, I., Kinalis, A., Nikoletseas, S.: An Adaptive Power Conservation Scheme for Heterogeneous Wireless Sensors. In: Proc. 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2005), ACM Press, pp. 96–105 (2005). Also in: Theory Comput Syst (TOCS) J **42**(1), 42–72 (2008)
6. Estrin, D., Govindan, R., Heidemann, J., Kumar, S.: Next Century Challenges: Scalable Coordination in Sensor Networks. In: Proc. 5th ACM/IEEE International Conference on Mobile Computing, MOBICOM'1999
7. Heinzelman, W.R., Chandrakasan, A., Balakrishnan, H.: Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In: Proc. 33rd Hawaii International Conference on System Sciences, HICSS'2000
8. Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In: Proc. 6th ACM/IEEE International Conference on Mobile Computing, MOBICOM'2000
9. Kahn, J.M., Katz, R.H., Pister, K.S.J.: Next Century Challenges: Mobile Networking for Smart Dust. In: Proc. 5th ACM/IEEE International Conference on Mobile Computing, pp. 271–278, Sept. 1999

10. Leone, P., Rolim, J., Nikoletseas, S.: An Adaptive Blind Algorithm for Energy Balanced Data Propagation in Wireless Sensor Networks. In: Proc. of the IEEE International Conference on Distributed Computing in Sensor Networks (DCOSS). Lecture Notes in Computer Science (LNCS), vol. 3267, pp. 35–48. Springer (2005)
11. Luo, J., Hubaux, J.-P.: Joint Mobility and Routing for Lifetime Elongation in Wireless Networks. In: Proc. 24th INFOCOM (2005)
12. Nikoletseas, S., Chatzigiannakis, I., Antoniou, A., Efthymiou, C., Kinalis, A., Mylonas, G.: Energy Efficient Protocols for Sensing Multiple Events in Smart Dust Networks. In: Proc. 37th Annual ACM/IEEE Simulation Symposium (ANSS'04), pp. 15–24, IEEE Computer Society Press (2004)
13. Triantafillou, P., Ntarmos, N., Nikoletseas, S., Spirakis, P.: NanoPeer Networks and P2P Worlds. In:Proc. 3rd IEEE International Conference on Peer-to-Peer Computing (P2P 2003), pp. 40–46, Sept. 2003