O

Oblivious Routing

2002; Räcke

Nikhil Bansal
IBM Research, IBM, Yorktown Heights, NY, USA

Keywords and Synonyms

Fixed path routing

Problem Definition

Consider a communication network, for example, the network of cities across the country connected by communication links. There are several sender-receiver pairs on this network that wish to communicate by sending traffic across the network. The problem deals with routing all the traffic across the network such that no link in the network is overly congested. That is, no link in the network should carry too much traffic relative to its capacity. The obliviousness refers to the requirement that the routes in the network must be designed without the knowledge of the actual traffic demands that arise in the network, i. e. the route for every sender-receiver pair stays fixed irrespective of how much traffic any pair chooses to send. Designing a good oblivious routing strategy is useful since it ensures that the network is robust to changes in the traffic pattern.

Notations

Let $G = (V, E)$ be an undirected graph with non-negative capacities $c(e)$ on edges $e \in E$. Suppose there are k source-destination pairs (s_i, t_i) for $i = 1, \ldots, k$ and let d_i denote the amount of flow (or demand) that pair i wishes to send from s_i to t_i. Given a routing of these flows on G, the congestion of an edge e is defined as $u(e)/c(e)$, the ratio of the total flow crossing edge e divided by its capacity. The congestion of the overall routing is defined as the maximum congestion over all edges. The congestion minimization problem is to find the routing that minimizes the maximum congestion. Observe that specifying a flow from s_i to t_i is equivalent to finding a probability distribution (not necessarily unique) on a collection of paths from s_i to t_i.

The congestion minimization problem can be studied in many settings. In the offline setting, the instance of the flow problem is provided in advance and the goal is to find the optimum routing. In the on line setting, the demands arrive in an arbitrary adversarial order and a flow must be specified for a demand immediately upon arrival; this flow is fixed forever and cannot be rerouted later when new demands arrive. Several distributed approaches have also been studied where each pair routes its flow in a distributed manner based on some global information such as the current congestion on the edges.

In this note, the oblivious setting is considered. Here a routing scheme is specified for each pair of vertices in advance without any knowledge of which demands will actually arrive. Note that an algorithm in the oblivious setting is severely restricted. In particular, if d_i units of demand arrive for pair (s_i, t_i), the algorithm must necessarily route this demand according to the pre-specified paths irrespective of the other demands or any other information such as congestion of other edges. Thus given a network graph G, the oblivious flows need to be computed just once. After this is done, the job of the routing algorithm is trivial; whenever a demand arrives, it simply routes it along the pre-computed path. An oblivious routing scheme is called c-competitive if for any collection of demands D, the maximum congestion of the oblivious routing is no more than c times the congestion of the optimum offline solution for D. Given this stringent requirement on the quality of oblivious routing, it is not a priori clear that any reasonable oblivious routing scheme should exist at all.

Key Results

Oblivious routing was first studied in the context of permutation routing where the demand pairs form a permutation and have unit value each. It was shown that any oblivious routing that specifies a single path (instead of a flow) between every two vertices must neces-

sarily perform badly. This was first shown by Borodin and Hopcroft [6] for hypercubes and the argument was later extended to general graphs by Kaklamanis, Krizanc and Tsantilas [12], who showed the following.

Theorem 1 ([6,12]) *For every graph G of size n and maximum degree d and every oblivious routing strategy using only a single path for every source-destination pair, there is a permutation that causes an overlap of at least $(n/d)^{1/2}$ paths at some node. Thus if each edge in G has unit capacity, the edge congestion is at least $(n/d)^{1/2}/d$.*

Since there exists constant degree graphs such as the butterfly graphs that can route any permutation with logarithmic congestion, this implies that such oblivious routing schemes must necessarily perform poorly on certain graphs.

Fortunately, the situation is substantially better if the single path requirement is relaxed and a probability distribution on paths (equivalently a flow) is allowed between each pair of vertices. In a seminal paper, Valiant and Brebner [17] gave the first oblivious permutation routing scheme with low congestion on the hypercube. It is instructive to consider their scheme. Consider an hypercube with $N = 2^n$ vertices. Represent vertex i by the binary expansion of i. For any two vertices s and t, there is a canonical path (of length at most $n = \log N$) from s to t obtained by starting from s and flipping the bits of s in left to right order to match with that of t. Consider routing scheme that for a pair s and t, it first chooses some node p uniformly at random, routes the flow from s to p along the canonical path, and then routes it again from p to t along the canonical path (or equivalently it sends $1/N$ units of flow from s to each intermediate vertex p and then routes it to t). An relatively simple analysis shows that

Theorem 2 ([17]) *The above oblivious routing scheme achieves a congestion of $O(1)$ for hypercubes.*

Subsequently, oblivious routing schemes were proposed for few other special classes of networks. However, the problem of designing oblivious routing schemes for general graphs remained open until recently, when in a breakthrough result Räcke showed the following.

Theorem 3 ([15]) *For any undirected capacitated graph $G = (V, E)$, there exist an oblivious routing scheme with congestion $O(\log^3 n)$ where n is the number of vertices in G.*

The key to Räcke's theorem is a hierarchical decomposition procedure of the underlying graph (described in further detail below). This hierarchical decomposition is a fundamental combinatorial result about the cut structure of graphs and has found several other applications, some of which are mentioned in Section "Applications". Räcke's proof of Theorem 3 only showed the existence of a good hierarchical decomposition and did not give an efficient polynomial time algorithm to find it. In subsequent work, Harrelson, Hildrum and Rao [11] gave a polynomial time procedure to find the decomposition and improved the competitive ratio of the oblivious routing to $O(\log^2 n \log\log n)$.

Theorem 4 ([11]) *There exists an $O(\log^2 n \log\log n)$-competitive oblivious routing scheme for general graphs and moreover it can be found in polynomial time.*

Interestingly, Azar et al. [4] show that the problem of finding the optimum oblivious routing for a graph can be formulated as a linear program. They consider a formulation with exponentially many constraints; one for each possible demand matrix that has optimum congestion 1, that enforces that the oblivious routing should have low congestion for this demand matrix. Azar et al. [4] give a separation oracle for this problem and hence it be solved using the ellipsoid method. A more practical polynomial size linear program was given later by Applegate and Cohen [2]. Bansal et al. [5] considered a more general variant referred to as the online oblivious routing that can also be used to find an optimum oblivious routing. However, note that without Räcke's result, it would not be clear whether these optimum routings were any good. Moreover these techniques do not give a hierarchical decomposition, and hence may be less desirable in certain contexts. On the other hand, they may be more useful sometimes since they produce an optimum routing (while [11] implies an $O(\log^2 n \log\log n)$-competitive routing for any graph, the best oblivious routing could have a much better guarantee for a specific graph).

Oblivious routing has also been studied for directed graphs, however the situation is much worse here. Azar et al. [4] show that there exist directed graphs where any oblivious routing is $\Omega(\sqrt{n})$ competitive. Some positive results are also known [10]. Hajiaghayi et al. [8] show a substantially improved guarantee of $O(\log^2 n)$ for directed graphs in the random demands model. Here each source-sink pair has a distribution (that is known by the algorithm) from which it chooses its demand independently. A relaxation of oblivious routing known as semi-oblivious routing has also been studied recently [9].

Techniques

This section describes the high level idea of Räcke's result. For a subset $S \subset V$, let $\text{cap}(S)$ denote the total capacity of the edges that cross the cut $(S, V \setminus S)$ and let $\text{dem}(S)$

denote the total demand that must be routed across the cut $(S, V \setminus S)$. Observe that $q = \max_{S \subset V} \text{dem}(S)/\text{cap}(S)$ is a lower bound on the congestion of any solution. On the other hand, the key result [3,13] relating multi-commodity flows and cuts implies that there is a routing such that the maximum congestion is at most $O(q \log k)$ where k is the number of distinct source sink pairs. However, note that this by itself does not suffice to obtain good oblivious routings, since a pair (s_i, t_i) can have different routing for different demand sets. The main idea of Räcke was to impose a tree like structure for routing on the graph to achieve obliviousness. This is formalized by a hierarchical decomposition described below.

Consider a hierarchical decomposition of the graph $G = (V, E)$ as follows. Starting from the set $S = V$, the sets are partitioned successively until each set becomes singleton vertex. This hierarchical decomposition can be viewed naturally as a tree T, where the root corresponds to the set V, and leaves corresponds to the singleton sets $\{v\}$. Let S_i denote the subset of V corresponding to node i in T. For an edge (i, j) in the tree where i is the child of j, assign it a capacity equal to $\text{cap}(S_i)$ (note that this is the capacity from S_i to the rest of G and not just capacity between S_i and S_j in G). The tree T is used to simulate routing in G and vice versa. Given a demand from u to v in G, consider the corresponding (unique) route among leaves corresponding to $\{u\}$ and $\{v\}$ in T. For any set of demands, it is easily seen that the congestion in T is no more than the congestion in G. Conversely, Räcke showed that there also exists a tree T where the routes in T can be mapped back to flows in G, such that for any set of demands the congestion in G is at most $O(\log^3 n)$ times that in T. In this mapping a flow along the (i, j) in the tree T corresponds to a suitably constructed flow between sets S_i and S_j in G. Since route between any two vertices in T is unique, this gives an oblivious routing in G.

Räcke uses very clever ideas to show the existence of such a hierarchical decomposition. Describing the construction is beyond the scope of this note, but it is instructive to understand the properties that must be satisfied by such a decomposition. First, the tree T should capture the bottlenecks in G, i. e. if there is a set of demands that produces high congestion in G, then it should also produce a high congestion in T. A natural approach to construct T would be to start with V, split V along a bottleneck (formally, along a cut with low sparsity), and recurse. However, this approach is too simple to work. As discussed below, T must also satisfy two other natural conditions, known as the *bandwidth* property and the *weight* property which are motivated as follows. Consider a node i connected to its parent j in T. Then, i needs to route $\text{dem}(S_i)$ flow out of S_i and it incurs congestion $\text{dem}(S_i)/\text{cap}(S_i)$ in T. However, when T is mapped back to G, all the flow going out of S_i must pass via S_j. To ensure that the edges from S_i to S_j are not overloaded, it must be the case that the capacity from S_i to S_j is not too small compared to the capacity from S_i to the rest of the graph $V \setminus S_i$. This is referred to as the bandwidth property. Räcke guarantees that this is ratio is always $\Omega(1/\log n)$ for every S_i and S_j corresponding to edges (i, j) in the tree. The weight property is motivated as follows. Consider a node j in T with children $i_1, \ldots, i_p$, then the weight property essentially requires that the sets $S_{i_1}, \ldots, S_{i_p}$ should be well connected among themselves even when restricted to the subgraph S_j. To see why this is needed, consider any communication between, say nodes i_1 and i_2 in T. It takes the route i_1 to j to i_2, and hence in G, S_{i_1} cannot use edges that lie outside S_j to communicate with S_{i_2}. Räcke shows that these conditions suffice and that a decomposition can be obtained that satisfies them.

The factor $O(\log^3 n)$ in Räcke's guarantee arises from three sources. The first logarithmic factor is due to the flow-cut gap [3,13]. The second is due to the logarithmic height of the tree, and the third is due to the loss of a logarithmic factor in the bandwidth and weight properties.

Applications

The problem has widespread applications to routing in networks. In practice it is often required that the routes must be a single path (instead of flows). This can often be achieved by randomized rounding techniques (sometimes under an assumption that the demands to capacity ratios be not too large). The flow formulation provides a much cleaner framework for studying the problems above.

Interestingly, the hierarchical decomposition also found widespread uses in other seemingly unrelated areas such as obtaining good pre-conditioners for solving systems of linear equations [14,16], for the all-or-nothing multicommodity flow problem [7], online network optimization [1] and so on.

Open Problems

It is possible that any graph has an $O(\log n)$ competitive oblivious routing scheme. Settling this is a key open question.

Cross References

- ► Routing
- ► Separators in Graphs
- ► Sparsest Cut

Recommended Reading

1. Alon, N., Awerbuch, B., Azar, Y., Buchbinder, N., Naor, J.: A general approach to online network optimization problems. In: Symposium on Discrete Algorithms, pp. 570–579 (2004)
2. Applegate, D., Cohen, E.: Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs. In: SIGCOMM, pp. 313–324 (2003)
3. Aumann, Y., Rabani, Y.: An O(log k) approximate min-cut max-flow theorem and approximation algorithm. SIAM J. Comput. **27**(1), 291–301 (1998)
4. Azar, Y., Cohen, E., Fiat, A., Kaplan, H., Räcke, H.: Optimal oblivious routing in polynomial time. In: Proceedings of the 35th ACM Symposium on the Theory of Computing, pp. 383–388 (2003)
5. Bansal, N., Blum, A., Chawla, S., Meyerson, A.: Online oblivious routing. In Symposium on Parallelism in Algorithms and Architectures, pp. 44–49 (2003)
6. Borodin, A., Hopcroft, J.: Routing, merging and sorting on parallel models of computation. J. Comput. Syst. Sci. **10**(1), 130–145 (1985)
7. Chekuri, C., Khanna, S., Shepherd, F.B.: The All-or-Nothing Multicommodity Flow Problem. In: Proceedings of 36th ACM Symposium on Theory of Computing, pp. 156–165 (2004)
8. Hajiaghayi, M., Kim, J.H., Leighton, T., Räcke, H.: Oblivious routing in directed graphs with random demands. In: Symposium on Theory of Computing, pp. 193–201 (2005)
9. Hajiaghayi, M., Kleinberg, R., Leighton, T.: Semi-oblivious routing: lower bounds. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms, pp. 929–938 (2007)
10. Hajiaghayi, M., Kleinberg, R., Leighton, T., Räcke, H.: Oblivious routing in node-capacitated and directed graphs. In: Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms, pp. 782–790 (2005)
11. Harrelson, C., Hildrum, K., Rao, S.: A polynomial-time tree decomposition to minimize congestion. In: Proceedings of the 15th annual ACM Symposium on Parallel Algorithms and Architectures, pp. 34–43 (2003)
12. Kaklamanis, C., Krizanc, D., Tsantilas, T.: Tight bounds for oblivious routing in the hypercube. In: Proceedings of the 3rd annual ACM Symposium on Parallel Algorithms and Architectures, pp. 31–36 (1991)
13. Linial, N., London, E., Rabinovich, Y.: The geometry of graphs and some of its algorithmic applications. In: IEEE Symposium on Foundations of Computer Science, pp. 577–591 (1994)
14. Maggs, B.M., Miller, G.L., Parekh, O., Ravi, R., Woo, S.L.M.: Finding effective support-tree preconditioners. In: Symposium on Parallel Algorithms and Architectures, pp. 176–185 (2005)
15. Räcke, H.: Minimizing congestion in general networks. In: Proceedings of the 43rd Annual Symposium on the Foundations of Computer Science, pp. 43–52 (2002)
16. Vaidya, P.: Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Unpublished manuscript (1991)
17. Valiant, L., Brebner, G.J.: Universal schemes for parallel communication. In: Proceedings of the 13th ACM Symposium on Theory of Computing, pp. 263–277 (1981)

Obstacle Avoidance Algorithms in Wireless Sensor Networks

2007; Powell, Nikoletseas

Sotiris Nikoletseas[1], Olivier Powell[2]

[1] Computer Engineering and Informatics Department, Computer Technology Institute, Patras, Greece

[2] Informatics Department, University of Geneva, Geneva, Switzerland

Keywords and Synonyms

Greedy geographic routing; Routing holes

Problem Definition

Wireless sensor networks are composed of many small devices called sensor nodes with sensing, computing and radio frequency communication capabilities. Sensor nodes are typically deployed in an ad hoc manner and use their sensors to collect environmental data. The emerging network collectively processes, aggregates and propagates data to regions of interest, e. g. from a region where an event is being detected to a base station or a mobile user. This entry is concerned with the data propagation duty of the sensor network in the presence of obstacles.

For different reasons, including energy conservation and limited transmission range of sensor nodes, information propagation is achieved via multi-hop message transmission, as opposed to single-hop long range transmission. As a consequence, message routing becomes necessary. Routing algorithms are usually situated at the network layer of the protocol stack where the most important component is the (dynamic) communication graph.

Definition 1 (Communication graph) A wireless sensor network is viewed as a graph $G = (V, E)$ where vertexes correspond to sensor nodes and edges represent wireless links between nodes.

Wireless sensor networks have stringent constraints that make classical routing algorithms inefficient, unreliable or even incorrect. Therefore, the specific requirements of wireless sensor networks have to be addressed [2] and geographic routing offers the possibility to design particularly well adapted algorithms.

Geographic Routing

A geographic routing algorithm takes advantage of the fact that sensor nodes are location aware, i. e. they know their position in a coordinate system following the use of a localization protocol [7]. Although likely to introduce a sig-

nificant overhead, the use of a localization protocol is also likely to be inevitable in many applications where environmental data collected by the sensors would be useless if not related to some geographical information. For those applications, node location awareness can be assumed to be available for routing purposes at no additional cost.

The Power of Simple Geographic Routing The early "most forward within range" (MFR) or greedy geographic routing algorithms [14] route messages by maximizing, at each hop, the progress on a projected line towards the destination or, alternatively, minimizing the remaining distance to the message's destination. Both of these greedy heuristics are referred to as *greedy forwarding* (GF). Greedy forwarding is a very appealing routing technique for wireless sensor networks. Among explanations for the attractiveness of GF are the following. (1) GF, as is almost imperatively required, is *fully distributed*. (2) It is *lightweight* in the sense that it induces no topology control overhead. (3) It is *all-to-all* (as opposed to all-to-one). (4) Making no assumptions on the structure of the communication graph, which can be directed, undirected, stable or dynamic (e. g. nodes may be mobile or wireless links may appear and disappear, for example following environmental fluctuation or as a consequence of lower protocol stack layers such as sleep/awake schemes for energy saving purposes), it is *robust*. (5) It is *on-demand*: no routing table or gradient has to be built prior to message propagation. (6) *Efficiency* is featured as messages find short paths to their destination in terms of hop count. (7) It is very *simple* and thus *easy to implement*. (8) It is *memory efficient* in the sense that (8a) the only information stored in the message header is the message's destination and that (8b) it is "ecologically sound" because no "polluting" information is stored on the sensor nodes visited by messages.

Problem Statement

Although very appealing, GF suffers from a major flaw: when a message reaches a local minimum where no further progress towards the destination is possible the routing algorithm fails. There are two major reasons for the occurrence of local minimums: routing holes [1] and obstacles.

Definition 2 The so called *routing holes* are low density regions of the network where no sensor nodes are available for next-hop forwarding.

Even in uniform-randomly deployed networks, routing holes appear as the manifestation of statistical variance of node density. Although increasing as network density diminishes, routing holes have a severe impact on the performance of GF even for very high density networks [12].

Definition 3 A *transmission blocking obstacle* is a region of the network where no sensors are deployed and through which radio signals do not propagate.

Clearly, large obstacles lying between a message and its destination tend to make GF fail.

The problem reported in this entry is to find a geographic routing algorithm that *maintains the advantages of greedy forwarding* listed in Sect. "Geographic Routing" such as simplicity, light weight, robustness and efficiency while *overcoming its weaknesses*: the inability to escape local minimum nodes created by routing holes and large transmission blocking obstacles such as those seen in Fig. 1.

Problem 1 (Escaping routing holes) *The first problem is to route messages out of the many routing holes which are statistically doomed to occur even in dense networks.*

Problem 2 (Contouring obstacles) *The second problem is to design a protocol capable of routing messages around large transmission blocking obstacles.*

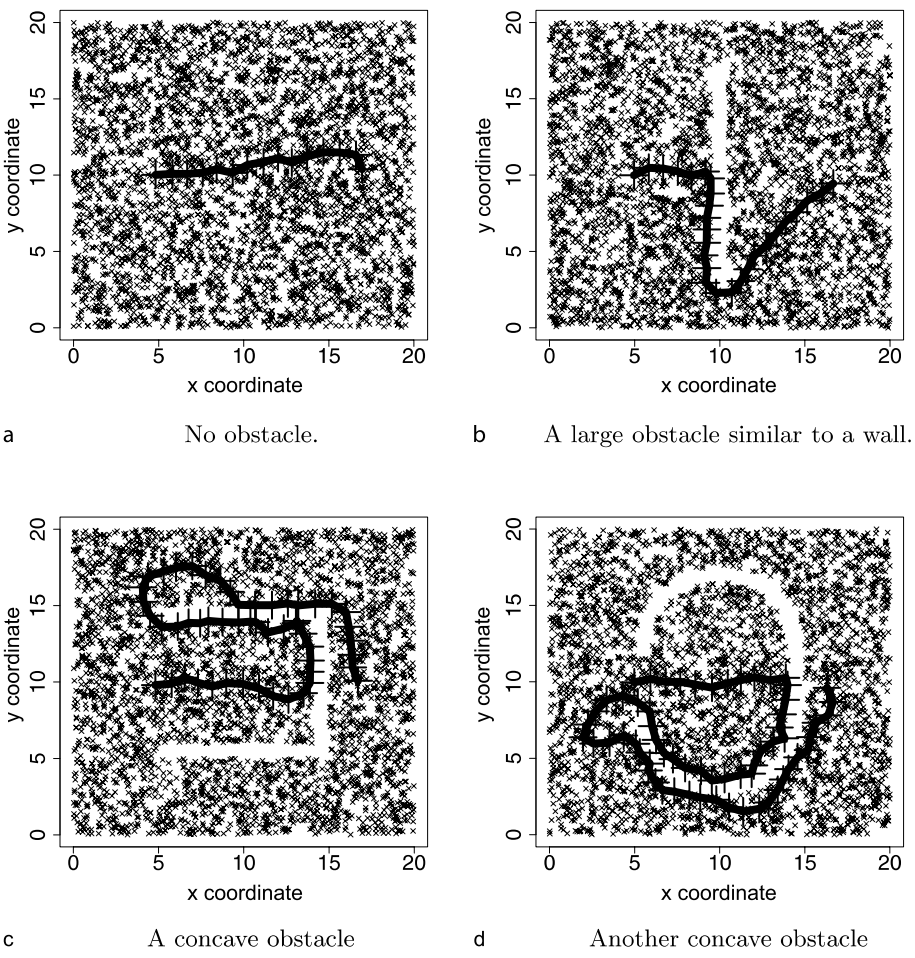
Problem 1 can be considered a simplified instance of problem 2. Lightweight solutions to problem 1 have been previously proposed, usually using limited backtracking [6] or controlled flooding combined with a GF heuristic [4,13]. However, as shown in [5] where an integrated model for obstacles is proposed and where different algorithms are compared with respect to their obstacle avoidance capability, those solutions do not satisfactorily solve problem 2 in the sense that only small and simple obstacles are efficiently bypassed.

Key Results

In [12] a new **g**eographic **r**out**i**ng around obsta**c**les (GRIC) algorithm was proposed to address the problems described in the previous section.

Basic Idea of the Algorithm

In GF, the strategy is to always propagate the message to the neighbor that maximizes progress *towards the destination*. Similarly, GRIC also maximizes progress in a chosen direction. However, this direction is not necessarily the message's destination but an *ideal direction of progress* which has to be computed according to one of two possible strategies: the *inertia mode* or the *rescue mode* described below. Finally, it was found that performance is better in the presence of slightly unstable networks, c. f.



Obstacle Avoidance Algorithms in Wireless Sensor Networks, Figure 1
Typical path followed by GRIC to bypass certain obstacles

Result 4, and thus in the case where the communication graph is very stable it is recommended to use a randomized version of GRIC where nodes about to take a routing decision randomly mark as either passive or active each outbound wireless link of the communication graph. Only active wireless links can be used for message propagation, and link status is re-evaluated each time a new routing decision is taken. Marking links as active with a probability of $p = 0.95$ was found to be a good choice for practical purposes [12].

Inertia Mode The idea of the inertia mode is that a message should have a strong incentive to go towards its destination but this incentive should be moderated by one to follow the straight ahead direction of current motion *"... like a celestial body in a planet system ..."* [12]. The inertia mode aims at making messages follow closely the perimeter of routing holes and obstacles in order to eventually bypass them and ensure final routing to the destination. To implement the inertia mode, a single additional assumption is made: sensor nodes should be aware of the position of the node from which they receive a message. As an example, this could be done by piggy-backing this 1-hop away routing path history in the message header. Knowing its own position p, the message's destination and the 1-hop away previous position of the message a sensor node can compute the vectors v_{cur} and v_{dst} starting at position p and pointing in the direction of current motion and the direction to the message's destination respectively. The inertia mode defines the ideal direction of progress, v_{idl}, as a vector starting at point p and lying "somewhere in between" v_{cur} and v_{dst}. More precisely, let α be the only angle in $[-\pi, \pi[$ such that v_{dst} is obtained by applying a rotation of angle α to v_{cur}, then v_{idl} is the vector ob-

tained by applying a rotation of angle α' to v_{cur}, where $\alpha' = \text{sign}(\alpha) \cdot \min\left\{\frac{\pi}{6}, |\alpha|\right\}$. Finally, the message is greedily forwarded to the neighbor node maximizing progress in the computed ideal direction of progress v_{idl}.

Rescue Mode In order to improve overall performance and to bypass complex obstacles, the rescue mode imitates the right-hand rule (RHR) which is a well known wall follower technique to find one's way out of a maze. A high-level description of the RHR component of GRIC is given below while details will be found in [12]. In GRIC, the RHR makes use of a virtual compass and a flag. The virtual compass assigns to v_{cur} a cardinal point value, treating the message's destination as the north. Considering the angle α defined in the previous section, the compass returns a string x-y with x equal to north or south if $|\alpha|$ is smaller or greater than $\frac{\pi}{2}$ respectively, while y is equal to west or east if α is negative or positive respectively. The first time the compass returns a south value, the flag is raised and tagged with the (x, y) value of the compass. Raising the flag means that the message is being routed around an obstacle using the RHR rule if the compass indicates south-west. In the case where the compass indicates south-east, a symmetric case not discussed here for brevity is applied using the left-hand rule (LHR) instead of the RHR. Once the flag is raised, it stays up with its tag unchanged until the compass indicates north, meaning that the obstacle has been bypassed. In fact, a small optimization can be made by lowering the flag only if the compass points to the north-west (in the case of the RHR) and not if it points north-east, but c.f. [12] for details. According to the RHR the obstacle's perimeter should be followed closely and kept on the right side of the message's current direction. If ever the compass and the flag's tag disagree, i. e. if the flag is tagged with south-west and the compass returns south-east, it is assumed that the message is turning left too much, that it risks going away from the obstacle and that the RHR is at risk of being violated (a symmetric case applies for the LHR). When this is so, GRIC responds by calling the rescue mode which changes the default way of computing v_{idl}: in rescue mode the message is forced to turn right (or left if the LHR is applied), by defining v_{idl} as the vector obtained by applying to v_{cur} a rotation of angle α'' (instead of α' in inertia mode) where $\alpha'' = -sign(\alpha)(2\pi - |\alpha|)/6$.

Main Findings

The performance of GRIC was evaluated through simulations. The main parameters were the presence (or absence) of different shapes of large communication blocking obstacles and the network density which ranged from very low to very high and controls the average degree of the communication graph and the occurrence of routing holes. The main performance metrics were the success rate, i. e. the percentage of messages routed to destination, and the path length. The main findings are that GRIC efficiently, i. e. using short paths, bypasses routing holes and obstacles but that in the presence of hard obstacles, the performance decreases with network density. In Figure 1, typical routing paths found by GRIC for different obstacle shapes are illustrated, c. f. [12] for details on the simulation environment.

Result 1 *In the absence of obstacles, routing holes are bypassed for every network density: The success rate is close to 100% as long as the source and the destination are connected. Also, routing is efficient in the sense that path lengths are very short.*

Result 2 *Some convex obstacles such as the one in Fig. 1b are bypassed with almost 100% success rate and using short paths, even for low densities. When the density gets very low performance diminishes: If the density gets below the critical level guaranteeing the communication graph to be connected with high probability, then the success probability diminishes quickly and successful routings use longer routing paths.*

Result 3 *Some large concave obstacles such as those in Fig. 1c and d are efficiently bypassed. However, when facing such obstacles performance becomes more sensitive to network density. The success rate drops and routing paths become longer when the density gets below a certain level depending on the exact obstacle shape.*

Result 4 (Robustness) *Similarly to* GF, GRIC *is robust to link instability. Furthermore, it was observed that limited link instability has a significantly positive impact on performances. This can be understood as the fact that messages are less likely to enter endless routing loops in a "hot" system than in a "cold" system.*

Applications

Replacement for Greedy Forwarding

Because it makes no compromise with the advantages of GF except the fact that it may be somehow more complicated to implement and because it overcomes GF's main limitations, GRIC can probably replace GF for most routing scenarios including but not exclusively wireless sensor networks. As an example opportunistic-routing strategies [11] could be applied to GRIC rather than to GF.

Wireless Sensor Networks with Large Obstacles

GRIC successfully bypasses large communication blocking obstacles. However, it does so efficiently only if the network density is high enough. This suggests that the obstacle avoidance feature of GRIC may be more useful for dense wireless networks than for sparse networks. Wireless sensor networks are an example of networks which are usually considered to be dense.

Dynamic Networks

There exist some powerful alternatives to GRIC such as the celebrated guaranteed delivery protocols GFG [3], GPSR [8] or GOAFR [10]. Those protocols rely on a planarization phase such as the lazy cross-link detection protocol (CLDP) [9]. LCR implies significant topology maintenance overhead which would be amortized over time if the network is stable enough. On the contrary, if the network is highly dynamic the necessity for frequent updates could make this topology maintenance overhead prohibitive. GRIC may thus be a preferable choice for dynamic networks where the communication graph is not a stable structure.

Open Problems

(1) Hard concave obstacles such as the one in Figure 1d are still a challenge for lightweight protocol since in this configuration GRIC's performance is strongly dependent on network density. (2) Low to very low densities are challenging when combined with large obstacles, even when they are "simple" convex obstacles like the one in Figure 1b. (3) The problem reported in this entry in the case of 3-dimensional networks is open. Inertia may be of some help, however the virtual compass and the right-hand rule seem quite strongly depend-ant on the 2-dimensional plane. (4) GRIC is not loop free. A mechanism to detect loops or excessively long routing paths would be quite important for practical purposes. (5) The understanding of GRIC could be improved. Analytical results are lacking and new metrics could be considered such as network lifetime, energy consumption or traffic congestion.

Cross References

► Probabilistic Data Forwarding in Wireless Sensor Networks

Recommended Reading

1. Ahmed, N., Kanhere, S.S., Jha, S.: The holes problem in wireless sensor networks: a survey. SIGMOBILE Mob. Comput. Commun. Rev. **9**, 4–18 (2005)
2. Al-Karaki, J.N., Kamal, A.E.: Routing techniques in wireless sensor networks: a survey. Wirel. Commun. IEEE **11**, 6–28 (2004)
3. Bose, P., Morin, P., Stojmenovic, I., Urrutia, J.: Routing with guaranteed delivery in ad hoc wireless networks. In: Discrete Algorithms and Methods for Mobile Computing and Communications (1999)
4. Chatzigiannakis, I., Dimitriou, T., Nikoletseas, S., Spirakis, P.: A probabilistic forwarding protocol for efficient data propagation in sensor networks. In: European Wireless Conference on Mobility and Wireless Systems beyond 3G (EW 2004), pp. 344–350. Barcelona, Spain, 27 February 2004
5. Chatzigiannakis, I., Mylonas, G., Nikoletseas, S.: Modeling and evaluation of the effect of obstacles on the performance of wireless sensor networks. In: 39th ACM/IEEE Simulation Symposium (ANSS), Los Alamitos, CA, USA, IEEE Computer Society, pp. 50–60 (2006)
6. Chatzigiannakis, I., Nikoletseas S., Spirakis, P.: Smart dust protocols for local detection and propagation. J. Mob. Netw. (MONET) **10**, 621–635 (2005)
7. Karl, H., Willig, A.: Protocols and Architectures for Wireless Sensor Networks. Wiley, West Sussex (2005)
8. Karp, B., Kung, H.T.: GPSR: greedy perimeter stateless routing for wireless networks. In: Mobile Computing and Networking. ACM, New York (2000)
9. Kim, Y.J., Govindan, R., Karp, B., Shenker, S.: Lazy cross-link removal for geographic routing. In: Embedded Networked Sensor Systems. ACM, New York (2006)
10. Kuhn, F., Wattenhofer, R., Zhang, Y., Zollinger, A.: Geometric ad-hoc routing: of theory and practice. In: Principles of Distributed Computing. ACM, New York (2003)
11. Lee, S., Bhattacharjee, B., Banerjee, S.: Efficient geographic routing in multihop wireless networks. In MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing, pp. 230–241. ACM, New York (2005)
12. Powell, O., Nikolesteas, S.: Simple and efficient geographic routing around obstacles for wireless sensor networks. In: WEA 6th Workshop on Experimental Algorithms, Rome, Italy. Springer, Berlin (2007)
13. Stojmenovic, I., Lin, X.: Loop-free hybrid single-path/flooding routing algorithms with guaranteed delivery for wireless networks. IEEE Trans. Paral. Distrib. Syst. **12**, 1023–1032 (2001)
14. Takagi, H., Kleinrock, L.: Optimal transmission ranges for randomly distributed packet radio terminals. Communications, IEEE Trans. [legacy, pre - 1988]. **32**, 246–257 (1984)

O(log log *n*)-competitive Binary Search Tree

2004; Demaine, Harmon, Iacono, Patrascu

Chengwen Chris Wang
Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA

Keywords and Synonyms

Tango

Problem Definition

Here is a precise definition of BST algorithms and their costs. This model is implied by most BST papers, and developed in detail by Wilber [22]. A static set of n keys is stored in the nodes of a binary tree. The keys are from a totally ordered universe, and they are stored in symmetric order. Each node has a pointer to its left child, to its right child, and to its parent. Also, each node may keep $o(\log n)$ bits of additional information but no additional pointers.

A BST algorithm is required to process a sequence of m accesses (without insertions or deletions), $S = s_1, s_2, s_3, s_4 \ldots s_m$. The ith access starts from the root and follows pointers until s_i is reached. The algorithm can update the fields in any node or rotate any edges that it touches along the way. The cost of the algorithm to execute an access sequence is defined to be the number of nodes touched plus the number of rotations.

Let A be any BST algorithm, define $A(S)$ as the cost of performing sequence S and $\mathrm{OPT}(S, T_0)$ as the minimum cost to perform the sequence S. An algorithm A is T-competitive if for all possible sequences S, $A(S) \leq T * \mathrm{OPT}(S, T_0) + O(m + n)$.

Since the number of rotation needed to change any binary tree of n keys into another one (with the same n keys) is at most $2n - 6$ [4,5,12,13,15]. It follows that $\mathrm{OPT}(S, T_0)$ differs from $\mathrm{OPT}(S, T_0')$ by at most $2n - 6$. Thus, if $m > n$, then the initial tree can only affect the constant factor.

Key Results

The *interleave bound* is a lower bound on $\mathrm{OPT}(S, T_0)$ that depends only on S. Consider any binary search tree P of all the elements in T_0. For each node y in P, define the *left side* of y includes all nodes in y's left subtree and y. And define the *right side* of y includes all nodes in y's right subtree. For each node y, label each access s_i in S by whether it is in the left or right side of y, ignoring all accesses not in y's subtree. Denote the number of times the label changes for y as $\mathrm{IB}(S, y)$. The *interleave* bound $\mathrm{IB}(S)$ is $\sum_y \mathrm{IB}(S, y)$.

Theorem 1 (Interleave Lower Bound [6,22]) $\mathrm{IB}(S)/2 - n$ *is a lower bound on* $\mathrm{OPT}(S, T_0)$.

Demaine et al observes that it is impossible to use this lower bound to improve the competitive ratio beyond $\Theta(\log \log n)$.

Theorem 2 (Tango is $O(\log \log n)$-competitive BST [6]) *The running time of Tango BST on a sequence S of m accesses is* $O((\mathrm{OPT}(S, T_0)) + n) * (1 + \log \log n))$.

Applications

Binary search tree (BST) is one of the oldest data structures in the history of computer science. It is frequently used to maintain an ordered set of data. In the last 40 years, many specialized binary search trees have been designed for specific applications. Almost every one of them supports access, insertion and deletion in worst-case $O(\log n)$ time on average for random sequences of access. This matches the best theoretically possible worst-case bound. For most of these data structures, a random sequence of m accesses will use $\Theta(m \log n)$ time.

While it is impossible to have better asymptotic performance for a random sequence of m accesses, many of the real world access sequences are not random. For instance, if the set of accesses are randomly drawn from a *small* subset of k element, it's possible to answer all the accesses in $O(m \log k)$ time. A notable binary search tree is Splay Tree. It is proved to perform well for many access patterns [2,3,8,14,16,17,18]. As a result, Sleator and Tarjan [14] conjectured that splay tree is $O(1)$-competitive to the optimal off-line BST. After more than 20 years, the conjecture remains an open problem.

Over the years, several restricted types of optimality have been proved. Many of these restrictions and usage patterns are based on real world applications. If each access is drawn independently at random from a fixed distribution, D, Knuth [11] constructed a BST based on D that is expected to run in optimal time up to a constant factor. Sleator and Tarjan [14] achieve the same bound without knowing D ahead of time. Other types includes key-independent optimality [10] and BST with free rotations [1].

In 2004, Demaine et al suggested searching for alternative BST algorithms that have small but non-constant competitive factors [6]. They proposed *Tango*, the first data structure proved to achieve a non-trivial competitive factor of $O(\log \log n)$. This is a major step toward developing a $O(1)$-competitive BST, and this line of research could potentially replace a large number of specialized BSTs.

Open Problems

Following this paper, several new $O(\log \log n)$-competitive BST have emerged [9,21]. A notable example is Multi–Splay Trees [21]. It generalizes the interleave bound to include insertions and deletions. Multi–Splay Trees also have many theorems analogous to Splay Trees [20,21], such as the access lemma and the working set theorem. Wang [21] conjectured that Multi-Splay Trees is $O(1)$-competitive, but it remains an open problem.

Returning to the original motivation for this research, the problem of finding an $o(\log \log n)$-competitive on-line

BST remains open. Several attempts have been made to improve the lower bound [6,7,22], but none of them have led to a lower competitive ratio. Even in the off-line model, the problem of finding an $O(1)$-competitive BST is difficult. The best known off-line constant competitive algorithm uses dynamic programming and requires exponential time.

Cross References

- ▶ B-trees
- ▶ Degree-Bounded Trees
- ▶ Dynamic Trees

Recommended Reading

Based on Wilber [22]'s lower bound, Tango [6] is the first $O(\log \log n)$-competitive binary search tree. Using many of the ideas in Tango and Link-cut Trees [14,19], Multi-Splay Trees [21] generalize the competitive framework to include insertion and deletion. The recommended readings are *Self-adjusting binary search trees* by Sleator and Tarjan, *Lower bounds for accessing binary search trees with rotations* by Wilber, *Dynamic Optimality - Almost* by Demaine, et al, and $O(\log \log n)$ *dynamic competitive binary search tree by Wang, et al.*

1. Blum, A., Chawla, S., Kalai, A.: Static optimality and dynamic search-optimality in lists and trees. Algorithmica **36**, 249–260 (2003)
2. Cole, R.: On the dynamic finger conjecture for splay trees II: The proof. SIAM J. Comput. **30**(1), 44–85 (2000)
3. Cole, R., Mishra, B., Schmidt, J., Siegel, A.: On the dynamic finger conjecture for splay trees I: Splay sorting log *n*-block sequences. SIAM J. Comput. **30**(1), 1–43 (2000)
4. Crane, C.A.: Linear lists and priority queues as balanced binary trees. Technical Report STAN-CS-72-259, Computer Science Dept., Stanford University (1972)
5. Culik II, K., Wood, D.: A note on some tree similarity measures. Inf. Process. Lett. **15**(1), 39–42 (1982)
6. Demaine, E.D., Harmon, D., Iacono, J., Patrascu, M.: Dynamic optimality —almost. SIAM J. Comput. **37**(1), 240–251 (2007)
7. Derryberry, J., Sleator, D.D., Wang, C.C.: A lower bound framework for binary search trees with rotations. Technical Report CMU-CS-05-187, Carnegie Mellon University (2005)
8. Elmasry, A.: On the sequential access theorem and deque conjecture for splay trees. Theor. Comput. Sci. **314**(3), 459–466 (2004)
9. Georgakopoulos, G.F.: How to splay for log log *n*-competitiveness. In: Proc. 4th Int'l Workshop on Experimental and Efficient Algorithms (WEA), pp. 570–579 (2005)
10. Iacono, J.: Key-independent optimality. Algorithmica **42**(1), 3–10 (2005)
11. Knuth, D.E.: Optimum binary search trees. Acta Informatica **1**, 14–25 (1971)
12. Luccio, F., Pagli, L.: On the upper bound on the rotation distance of binary trees. Inf. Process. Lett. **31**(2), 57–60 (1989)
13. Mäkinen, E.: On the rotation distance of binary trees. Inf. Process. Lett. **26**(5), 271–272 (1988)
14. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM **32**(3), 652–686 (1985)
15. Sleator, D.D., Tarjan, R.E., Thurston, W.P.: Rotation distance, triangulations, and hyperbolic geometry. In: Proceedings 18th ACM Symposium on Theory of Computing (STOC), Berkeley, 1986, pp. 122–135
16. Sundar, R.: Twists, turns, cascades, deque conjecture, and scanning theorem. In: Proceedings 30th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 555–559 (1989)
17. Sundar, R.: On the deque conjecture for the splay algorithm. Combinatorica **12**(1), 95–124 (1992)
18. Tarjan, R.: Sequential access in play trees takes linear time. Combinatorica **5**(4), 367–378 (1985)
19. Tarjan, R.E.: Data structures and network algorithms, *CBMS-NSF Reg. Conf. Ser. Appl. Math.*, vol. 44. SIAM, Philadelphia, PA (1983)
20. Wang, C.C.: Multi-splay trees. Ph.D. Thesis, Carnegie Mellon University (2006)
21. Wang, C.C., Derryberry, J., Sleator, D.D.: *O*(log log *n*)-competitive dynamic binary search trees. In: Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Miami, 2006, pp. 374–383
22. Wilber, R.: Lower bounds for accessing binary search trees with rotations. SIAM J. Comput. **18**(1), 56–67 (1989)

Online Interval Coloring

1981; Kierstead, Trotter

Leah Epstein
Department of Math, University of Haifa, Haifa, Israel

Keywords and Synonyms

An extremal problem in recursive combinatorics

Problem Definition

Online interval coloring is a graph coloring problem. In such problems the vertices of a graph are presented one by one. Each vertex is presented in turn, along with a list of its edges in the graph, which are incident to previously presented vertices. The goal is to assign colors (which without loss of generality are assumed to be non-negative integers) to the vertices, so that two vertices which share an edge receive different colors, and the total number of colors used (or alternatively, the largest index of any color that is used) is minimized. The smallest number of colors, for which the graph still admits a valid coloring, is called the chromatic number of the graph.

The interval coloring problem is defined as follows. Intervals on the real line are presented one by one, and the online algorithm must assign each interval a color before

the next interval arrives, so that no two intersecting intervals receive the same color. The goal is again to minimize the number of colored used to color any interval. The last problem is equivalent to coloring of interval graphs. These are graphs which have a representation (or realization) where each interval represents a vertex, and two vertices share an edge if and only if they intersect. It is assumed that the interval graph arrives online together with its realization.

Given an interval graph, denote the size of the largest cardinality clique (complete subgraph) in it by ω. Interval graphs have the special property that in a realization, the set of vertices in a clique have a common point in which they all intersect.

Before discussing the online problem, some properties of interval graphs need to be stated. There exists a simple offline algorithm which produces an optimal coloring of interval graphs. An algorithm applies First Fit, if each time it needs to assign a color to an interval, it assigns a smallest index color which still produces a valid coloring. The optimal algorithm simply considers intervals sorted from left to right by their left endpoints, and applies First Fit. Note that the resulting coloring never uses more than ω colors. Indeed, interval graphs are perfect[1].

However, once intervals arrive online in an arbitrary order, it is impossible to design an optimal coloring. Consider a simple example where the two intervals [1,3] and [6,8] are introduced. If they are colored using two distinct colors, this is already sub-optimal, since using the same color for both of them is possible. However, if the sequence of intervals is augmented with [2,5] and [4,7], these two new intervals cannot receive the color of the previous intervals, or the same color for both new intervals. Thus three colors are used, even though a valid coloring using two colors can be designed. Note that even if it is known in advance that the input can be colored using exactly two colors, not knowing whether the additional intervals are as defined above, or alternatively, a single interval [2,7] arrives instead, leads to the usage of three colors instead of only two.

Online coloring is typically hard, which already applies to some simple graph classes such as trees. This is due to the lower bound of $\Omega(\log n)$, given by Gyárfás and Lehel [9] on the competitive ratio of online coloring of trees. There are very few classes for which constant bounds are known. One such class is line graphs, for which Bar-Noy, Motwani and Naor [3] showed that First-Fit is 2-competitive (specifically it uses at most $2 \cdot opt - 1$ colors, where OPT is the number of colors in an optimal coloring), and this is best possible. This result was later generalized to $k \cdot opt - k + 1$ for $(k+1)$-claw free graphs by [6] (note that line graphs are 3-claw free).

[1] A graph G is perfect if any induced subgraph of G, G' (including G), can be colored using $\omega(G')$ colors, where $\omega(G')$ is the size of the largest cardinality clique in G'. (For any graph, ω is a clear lower bound on its chromatic number).

Key Results

The paper of Kierstead and Trotter [11] provides a solution to the online interval coloring problem. They show that the best possible competitive ratio is 3 which is achieved by an algorithm they design. More accurately, the following theorem is proved in the paper.

Theorem 1 *Given an interval graph which is introduced online, and presented via its realization, any online algorithm uses at least $3\omega - 2$ colors to color the graph, and there exists an algorithm which achieves this bound.*

In the sequel the algorithm and the lower bound construction are described. Note that the algorithm does not need to know ω in advance. Moreover, even though the algorithm is deterministic, it was shown in [12] that the lower bound of 3 on the competitive ratio of online algorithms for interval coloring holds for randomized algorithms as well. Thus [11], gives a complete solution for the problem!

The main idea of the algorithm is creation of "levels". At the time of arrival of an interval, it is classified into a level as follows. Denote by A_k the union of sets of intervals which currently belong to all levels $1, \ldots, k$. Intervals are classified so that the largest cardinality clique in A_k is of size k. Thus, A_1 is simply a set of non-intersecting intervals. On arrival of an interval, the algorithm finds the smallest k such that the new interval can join level k, without violating the rule above. It can be shown that each level can be colored using two colors by an offline algorithm. Since the algorithm defined here is online, such a coloring cannot be found in general (see example above). However it is shown in [11] that at most three colors are required for each such level, and a coloring using three colors can be found by applying First Fit on each level (with disjoint sets of colors). Moreover, the first level can always colored using a single color, and ω is equal exactly to the number of levels. Thus a total number of colors, which is at most $3\omega - 2$, is used.

Next, the deterministic lower bound is sketched. The idea is to create levels as in the algorithm. The levels are called phases. Each phase increases the largest clique size by 1, until the value ω is reached. Moreover, every phase (except for the first one) increases the number of colors used by the algorithm by 3.

After each phase was created, some parts of the line are shrunk into single points. Given a point p, that is a result of shrinking an interval $[a, b]$. Every interval presented in the past which is contained in $[a, b]$ is also shrunk into p and therefore such a point inherits a list of colors which no interval that contains it can receive. This is done for simplicity of the proof and means that for a given point p that is the result of shrinking, either contains all intervals that were shrunk into this point, or it has no overlap with any of them.

The sequence construction stops once $3\omega - 2$ colors have been used. Therefore it can be assumed that an initial palette of $3\omega-3$ colors is given, where these colored can all be used by the algorithm. The ith color ever used is called color number i. As soon as color $3\omega - 2$ is used (even if it is before the construction is completed), the construction stops. The constructed input has a maximum clique is of size (at most) ω.

The sequence starts with introducing a large enough number of intervals N, this is phase 0. Since the algorithm is using at most $3\omega - 3$ colors, this means that there exists a set of $N/(3\omega-3)$ intervals that share the exact same color. All intervals are shrunk into single points. Later phases result in additional points.

Next, phase $i < \omega$ is defined. The phases are constructed in a way that in the beginning of phase i there is a large enough number of points that contain a given set of $3i - 2$ colors (points of interest). Without loss of generality, assume that these are colors $1 \ldots, 3i - 2$ where the size of the largest clique is i. There exist some other points containing other sets of i colors, or sets of at most $i - 1$ colors. All these points are called void points. At this time, the points of interest are partitioned into consecutive sets of four.

Next, some additional intervals are defined, increasing the size of largest clique by exactly one. Given a set of four points a_1, a_2, a_3, a_4, let b be the leftmost void point on the right hand side of a_1, between a_1 and a_2. If no such point exists, then let $b = (a_1 + a_2)/2$. Similarly, let c be the rightmost void point between a_3 and a_4, and if no such point exists then $c = (a_3 + a_4)/2$. Let d be a point between a_2 and a_3 that is not a void point. The intervals $I_1 = [a_1, (a_1+b)/2]$ and $I_2 = [(c + a_4)/2, a_4]$ are introduced. Clearly none of them may receive one of the currently used $3i - 2$ colors. If they both receive the same new color, the intervals $I_3 = [(a_1 + b)/2, d]$ and $I_4 = [d, (c + a_4)/2]$ are introduced. The interval I_3 intersects with a_2, and with I_1. Therefore it receives an additional color. The second interval I_4 intersects I_3, a_3 and I_2. Therefore a third new color is given to it. If I_1, I_2 receive distinct new colors, the interval $I_5 = [(a_1+b)/2, (c+a_4)/2]$ is introduced. Since I_5 intersects with I_1, I_2, a_2, a_3, it must get a third new color. Every such interval $[a_1, a_4]$ is shrunk into a single point containing $3i + 1$ colors. Since there are less than 3ω colors, and each point uses exactly $3i + 1 < 3\omega$ of them, there are less than $(3\omega)!$ such choices, and a large enough number of points, having the same set of colors, can be picked. The points containing this exact set of colors become the points of interest of the next phase, and the others become void points of the next phase. Points that are void points of previous phases and are not contained in shrunk intervals remain void points. The only points where the new intervals intersect are points with no previous intervals, and therefore the clique size increases by 1 exactly.

At this time phase $i + 1$ can be performed. After phase $\omega - 1$, there are at least $3\omega - 2$ colors in use and the claim is proved. Note that prior to that phase, a minimum number of four points of interest is required.

Applications

In this section, both real-world applications of the problem, and applications of the methods of Kierstead and Trotter [11] to related problems, are discussed.

Many applications arise in various communication networks. The need for connectivity all over the world is rapidly increasing. On the other hand, networks are still composed of very expensive parts. Thus application of optimization algorithms is required in order to save costs.

Consider a network with a line topology that consists of links. Each connection request is for a path between two nodes in the network. The set of requests assigned to a channel must consist of disjoint paths. The goal is to minimize the number of channels (colors) used. A connection request from a to b corresponds to an interval $[a, b]$ and the goal is to minimize the number of required channels to serve all requests.

Another network related application is that if the requests have constant duration c, and all requests have to be served as fast as possible. In this case the colors correspond to time slots, and the total number of colors corresponds to the schedule length.

These are just sample applications, the problem can be described as a scheduling problem as well, and it is clearly of theoretical interest being a natural online graph coloring problem.

Two later studies are of possible interest here, both due to their relevance to the original problem and for the usage of related methods.

The applications in networks stated above raise a generalized problem studied in the recent years. In these applications, it is assumed that once a connection request be-

tween two points is satisfied, the channel is blocked at least for the duration of this request. An interesting question, that was raised by Adamy and Erlebach [1], is the following. Assume that a request consists not only of a requested interval, but also from a bandwidth requirement. That is, a customer of a communication channel specifies exactly how much of the channel is needed. Thus, in some cases it is possible to have overlapping requests sharing the same channel. It is required that at every point, the sum of all bandwidth requirements of requests sharing a color cannot exceed the value 1, which is the capacity of the channel. This problem is called *online interval coloring with bandwidth*. In the paper [1], a (large) constant competitive algorithm was designed for the problem. The original interval coloring problem is a special case of this problem where all bandwidth requests are 1. Note that this problem is a generalization of *bin packing* as well, since bin packing is the special case of the problem where all requests have a common point. Azar et al. [2] designed an algorithm of competitive ratio of at most 10 for this problem. This was done by partitioning the requests into four classes based on their bandwidth requirements, and coloring each such class separately. The class of requests with bandwidth in $\left(\frac{1}{2}, 1\right]$ was colored using the basic algorithm of [11], since no two such requests colored with one color can overlap. The two other classes, which are $\left(0, \frac{1}{4}\right]$ and $\left(\frac{1}{4}, \frac{1}{2}\right]$ were colored using adaptations of the algorithm of [11]. Epstein and Levy [7,8] designed improved lower bounds on the competitive ratio, showing that online interval coloring with bandwidth is harder than online interval coloring.

Another problem related to coloring is the *max coloring* problem. In this problem each interval is given a nonnegative weight. Given a coloring, the weight of a color is the maximum weight of any vertex of this color. The goal is to minimize the sum of weights of the used colors. Note that if all weights are 1, max coloring reduces to the graph coloring problem. Pemmaraju, Raman and Varadarajan [13] studied the *offline* max coloring problem for interval graphs. They apply an algorithm which is based on the algorithm of [11]. Thus, they sort the intervals according to their weights, in a monotone nonincreasing order. However, since their algorithm is not online, they exploit the property stated above, that every level is actually 2-colorable, and thus this results in an offline 2-approximation to *max coloring* on interval graphs.

Epstein and Levin [5] studied online max coloring on interval graphs. They design a general reduction which allows to convert algorithms for graph coloring into algorithms for max coloring. The loss in the competitive ratio is a multiplicative factor of 4 for deterministic algorithms, and a factor of $e \approx 2.718$ for randomized algorithms. Thus, using the algorithm of [11] as a black box, they obtained a 12-competitive deterministic algorithm and a $3 \cdot e \approx 8.155$-competitive randomized algorithm. Another result of [5] is lower bound of 4 on the competitive ratio of any randomized algorithm.

Open Problems

Since the paper [11] provided a nice and clean solution to the online interval coloring problem, it does not directly raise open problems. Yet, one related problem is of interest to researchers over the last thirty years, which is the performance of First Fit on this problem. It was shown by Kierstead [10] that First Fit uses at most 40ω colors, thus implying that First Fit has a constant competitive ratio. The quest after the exact competitive ratio was never completed. The best current published results are an upper bound of $10k$ by [13] and a lower bound of $4.4k$ by Chrobak and Slusarek [4]. See [14] for recent developments. It is interesting to note that for online interval coloring with bandwidth, First Fit has an unbounded competitive ratio [1].

Another open problem is to find the best possible competitive ratios for online interval coloring with bandwidth and for max coloring of interval graphs. As stated above, the gap for coloring with bandwidth is currently between $24/7 \approx 3.4286$ by [7,8] and 10 [2], and the gap for max coloring is between 4 and 12 [5].

Recommended Reading

1. Adamy, U., Erlebach, T.: Online coloring of intervals with bandwidth. In: Proc. of the First International Workshop on Approximation and Online Algorithms (WAOA2003), pp. 1–12 (2003)
2. Azar, Y., Fiat, A., Levy, M., Narayanaswamy, N.S.: An improved algorithm for online coloring of intervals with bandwidth. Theor. Comput. Sci. **363**(1), 18–27 (2006)
3. Bar-Noy, A., Motwani, R., Naor, J.: The greedy algorithm is optimal for on-line edge coloring. Inf. Proc. Lett. **44**(5), 251–253 (1992)
4. Chrobak, M., Ślusarek, M.: On some packing problems relating to dynamical storage allocation. RAIRO J. Inf. Theor. Appl. **22**, 487–499 (1988)
5. Epstein, L., Levin, A.: On the max coloring problem. In: Proc. of the Fifth International Workshop on Approximation and Online Algorithms (WAOA2007) (2007), pp. 142–155
6. Epstein, L., Levin, A., Woeginger, G.J.: Graph coloring with rejection. In: Proc. of 14th European Symposium on Algorithms (ESA2006), pp. 364–375. (2006)
7. Epstein, L., Levy, M.: Online interval coloring and variants. In: Proc. of The 32nd International Colloquium on Automata, Languages and Programming (ICALP2005), pp. 602–613. (2005)
8. Epstein, L., Levy, M.: Online interval coloring with packing constraints. In: Proc. of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS2005), pp. 295–307. (2005)

9. Gyárfás, A., Lehel, J.: Effective on-line coloring of P_5-free graphs. Combinatorica **11**(2), 181–184 (1991)
10. Kierstead, H.A.: The linearity of first-fit coloring of interval graphs. SIAM J. Discret. Math. **1**(4), 526–530 (1988)
11. Kierstead, H.A., Trotter, W.T.: An extremal problem in recursive combinatorics. Congr. Numerantium **33**, 143–153 (1981)
12. Leonardi, S., Vitaletti, A.: Randomized lower bounds for online path coloring. In: Proc. of the second International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM'98), pp. 232–247. (1998)
13. Pemmaraju, S., Raman, R., Varadarajan, K.: Buffer minimization using max-coloring. In: Proc. of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004), pp. 562–571. (2004)
14. Trotter, W.T.: Current research problems: First Fit colorings of interval graphs. http://www.math.gatech.edu/~trotter/rprob.htm Access date: December 24, 2007.

Online Learning

► Perceptron Algorithm

Online List Update

1985; Sleator, Tarjan

Susanne Albers
Institute for Computer Science, University of Freiburg, Freiburg, Germany

Keywords and Synonyms

Self organizing lists

Problem Definition

The *list update problem* represents a classical online problem and, beside paging, is the first problem that was studied with respect to competitiveness. The list update problem is to maintain a dictionary as an unsorted linear list. Consider a set of items that is represented as a linear linked list. The system is presented with a request sequence σ, where each request is one of the following operations. (1) It can be an *access* to an item in the list, (2) it can be an *insertion* of a new item into the list, or (3) it can be a *deletion* of an item. To access an item, a list update algorithm starts at the front of the list and searches linearly through the items until the desired item is found. To insert a new item, the algorithm first scans the entire list to verify that the item is not already present and then inserts the item at the end of the list. To delete an item, the algorithm scans the list to search for the item and then deletes it.

In serving requests a list update algorithm incurs cost. If a request is an access or a delete operation, then the incurred cost is i, where i is the position of the requested item in the list. If the request is an insertion, then the cost is $n + 1$, where n is the number of items in the list before the insertion. While processing a request sequence, a list update algorithm may rearrange the list. Immediately after an access or insertion, the requested item may be moved at no extra cost to any position closer to the front of the list. These exchanges are called *free exchanges*. Using free exchanges, the algorithm can lower the cost on subsequent requests. At any time two adjacent items in the list may be exchanged at a cost of 1. These exchanges are called *paid exchanges*. The goal is to serve the request sequence so that the total cost is as small as possible.

Of particular interest are list update algorithms that work *online*, i. e. each request is served without knowledge of any future requests. The performance of online algorithms is usually evaluated using *competitive analysis*. Here an online strategy is compared to an optimal offline algorithm that knows the entire request sequence in advance and can serve it with minimum cost. Given a request sequence σ, let $A(\sigma)$ denote the cost incurred by an online algorithm A in serving σ, and let $OPT(\sigma)$ denote the cost incurred by an optimal offline algorithm OPT. Online algorithm A is called c-competitive if there is a constant α such that for all size lists and all request sequences σ, $A(\sigma) \leq c \cdot OPT(\sigma) + \alpha$. The factor c is also called the *competitive ratio*. The competitiveness must hold *for all size lists*.

Key Results

There are three well-known deterministic online algorithms for the list update problem.

Algorithm Move-To-Front: Move the requested item to the front of the list.

Algorithm Transpose: Exchange the requested item with the immediately preceding item in the list.

Algorithm Frequency-Count: Maintain a frequency count for each item in the list. Whenever an item is requested, increase its count by 1. Maintain the list so that the items always occur in nonincreasing order of frequency count.

The formulations of list update algorithms generally assume that a request sequence consists of accesses only. It is obvious how to extend the algorithms so that they can also handle insertions and deletions. On an insertion, the algorithm first appends the new item at the end of the list and then executes the same steps as if the item was re-

quested for the first time. On a deletion, the algorithm first searches for the item and then just removes it.

First consider the algorithms Move-To-Front, Transpose and Frequency-Count. Note that Move-To-Front and Transpose are *memoryless* strategies, i. e. they do not need any extra memory to decide where a requested item should be moved. Thus, from a practical point of view, they are more attractive than Frequency-Count. Sleator and Tarjan [16] analyzed the competitive ratios of the three algorithms.

Theorem 1 ([16]) *The Move-To-Front algorithm is 2-competitive.*

The algorithms Transpose and Frequency-Count are not c-competitive, for any constant c.

Karp and Raghavan [13] developed a lower bound on the competitiveness that can be achieved by deterministic online algorithms. This lower bound implies that Move-To-Front has an optimal competitive ratio.

Theorem 2 ([13]) *Let A be a deterministic online algorithm for the list update problem. If A is c-competitive, then $c \geq 2$.*

An interesting issue is randomization in the list update problem. Against adaptive adversaries, no randomized online algorithm for list update can be better than 2-competitive, see [6,14]. Thus one concentrates on algorithms against oblivious adversaries. Many randomized algorithms for list update have been proposed [1,2,12,14]. The following paragraphs describe the two most important algorithms. Reingold et al. [14] gave a very simple algorithm, called Bit.

Algorithm Bit: Each item in the list maintains a bit that is complemented whenever the item is accessed. If an access causes a bit to change to 1, then the requested item is moved to the front of the list. Otherwise the list remains unchanged. The bits of the items are initialized independently and uniformly at random.

Theorem 3 ([14]) *The Bit algorithm is 1.75-competitive against any oblivious adversary.*

Reingold et al. [14] generalized the Bit algorithm and proved an upper bound of $\sqrt{3} \approx 1.73$ against oblivious adversaries. The best randomized algorithm currently known is a combination of the Bit algorithm and a deterministic 2-competitive online algorithm called Timestamp proposed in [1].

Algorithm Timestamp (TS): Insert the requested item, say x, in front of the first item in the list that precedes x and that has been requested at most once since the last request to x. If there is no such item or if x has not been requested so far, then leave the position of x unchanged.

As an example, consider a list of six items being in the order $L: x_1 \to x_2 \to x_3 \to x_4 \to x_5 \to x_6$. Suppose that algorithm TS has to serve the second request to x_5 in the request sequence $\sigma = \ldots x_5, x_2, x_2, x_3, x_1, x_1, x_5$. Items x_3 and x_4 were requested at most once since the last request to x_5, whereas x_1 and x_2 were both requested twice. Thus, TS will insert x_5 immediately in front of x_3 in the list. A combination of Bit and TS was proposed by [3].

Algorithm Combination: With probability $\frac{4}{5}$ the algorithm serves a request sequence using Bit, and with probability $\frac{1}{5}$ it serves a request sequence using TS.

Combination achieves the best competitive ratio currently known.

Theorem 4 ([2]) *The algorithm Combination is 1.6-competitive against any oblivious adversary.*

Ambühl et al. [4] presented a lower bound for randomized list update algorithms.

Theorem 5 ([4]) *Let A be a randomized online algorithm for the list update problem. If A is c-competitive against any oblivious adversary, then $c \geq 1.50084$.*

Using techniques from learning theory, Blum et al. [9] recently gave a randomized online algorithm that, for any $\epsilon > 0$, is $(1.6+\epsilon)$-competitive and at the same time $(1+\epsilon)$-competitive against an offline algorithm that is restricted to serving a request sequence with a static list.

So far this entry has considered online algorithms. In the offline variant of the list update problem, the entire request sequence σ is known in advance. Ambühl [3] showed that the offline variant is NP-hard.

Reingold et al. [14] studied an extended cost model, called the P^d model, for the list update problem. In the P^d model there are no free exchanges and each paid exchange costs d. Reingold et al. analyzed deterministic and randomized strategies in this model.

Many of the concepts shown for self-organizing linear lists can be extended to binary search trees. The most popular version of self-organizing binary search trees are the *splay trees* presented by Sleator and Tarjan and the reader is refered to [17].

Regarding the history of the list update problem, prior to competitive analysis, list update algorithms were studied in scenarios where request sequences are generated according to probability distributions. The asymptotic expected cost incurred by an online algorithm in serving a single request was compared to corresponding cost incurred by the optimal static ordering. There exists a large body of literature. Chung et al. [11] showed that, for any distribution, the asymptotic service cost of Move-To-Front is at most $\pi/2$ times that of the optimal ordering.

This bound is tight. Rivest [15] identified distributions on which Transpose performs better than Move-To-Front.

Applications

Linear lists are one possibility for representing a set of items. Certainly, there are other data structures such as balanced search trees or hash tables that, depending on the given application, can maintain a set in a more efficient way. In general, linear lists are useful when the set is small and consists of only a few dozen items. The most important application of list update algorithms are locally adaptive data compression schemes. In fact, Burrows and Wheeler [10] developed a data compression scheme using linear lists that achieves a better compression than Ziv-Lempel based algorithms. Before the description of that algorithm, the next paragraph first presents a data compression scheme given by Bentley et al. [8] that is very simple and easy to implement.

In data compression one is given a string S that shall be compressed, i. e. that shall be represented using fewer bits. The string S consists of symbols, where each symbol is an element of the alphabet $\Sigma = \{x_1, \ldots, x_n\}$. The idea of data compression schemes using linear lists it to convert the string S of symbols into a string I of integers. An encoder maintains a linear list of symbols contained in Σ and reads the symbols in the string S. Whenever the symbol x_i has to be compressed, the encoder looks up the current position of x_i in the linear list, outputs this position and updates the list using a list update rule. If symbols to be compressed are moved closer to the front of the list, then frequently occurring symbols can be encoded with small integers. A decoder that receives I and has to recover the original string S also maintains a linear list of symbols. For each integer j it reads from I, it looks up the symbol that is currently stored at position j. Then the decoder updates the list using the same list update rule as the encoder. As list update rule one may use any (deterministic) online algorithm. Clearly, when the string I is actually stored or transmitted, each integer in the string should be coded using a variable length prefix code.

Burrows and Wheeler [10] developed a very effective data compression algorithm using self-organizing lists. The algorithm first applies a reversible transformation to the string S. The purpose of this transformation is to group together instances of a symbol x_i occurring in S. The resulting string S' is then encoded using the Move-To-Front algorithm. More precisely, the transformed string S' is computed as follows. Let m be the length of S. The algorithm first computes the m rotations (cyclic shifts) of S and sorts them lexicographically. Then it extracts the last character of these rotations. The kth symbol of S' is the last symbol of the kth sorted rotation. The algorithm also computes the index J of the original string S in the sorted list of rotations. Burrows and Wheeler gave an efficient algorithm to recover the original string S given only S' and J. The corresponding paper [10] gives a very detailed description of the algorithm and reports of experimental results. On the Calgary Compression Corpus [18], the algorithm outperforms the UNIX utilities compress and gzip and the improvement is 13% and 6%, respectively.

Open Problems

The most important open problem is to determine tight upper and lower bounds on the competitive ratio that can be achieved by randomized online list update algorithms against oblivious adversaries. It is not clear what the true competitiveness is. It is conjectured that the bound is below 1.6. However, as implied by Theorem 5 the performance ratio must be above 1.5.

Experimental Results

Early experimental analyses of the algorithms Move-To-Front, Transpose and Frequency Count were performed by Rivest [15] as well as Bentley and McGeoach [7]. A more recent and extensive experimental study was presented by Bachrach et al. [5]. They implemented and tested a large number of online list update algorithms on request sequences generated by probability distributions and Markov chains as well as on sequences extracted from the Calgary Corpus. It shows that the locality of reference considerably influences the absolute and relative performance of the algorithms. Bachrach et al. also analyzed the various algorithms as data compression strategies.

Recommended Reading

1. Albers, S.: Improved randomized on-line algorithms for the list update problem. SIAM J. Comput. **27**, 670–681 (1998)
2. Albers, S., von Stengel, B., Werchner, R.: A combined BIT and TIMESTAMP algorithm for the list update problem. Inf. Proc. Lett. **56**, 135–139 (1995)
3. Ambühl, C.: Offline list update is NP-hard. In: Proc. 8th Annual European Symposium on Algorithms, pp. 42–51. LNCS, vol. 1879. Springer (2001)
4. Ambühl, C., Gärtner, B., von Stengel, B.: Towards new lower bounds for the list update problem. Theor. Comput. Sci **68**, 3–16 (2001)
5. Bachrach, B., El-Yaniv, R., Reinstädtler, M.: On the competitive theory and practice of online list accessing algorithms. Algorithmica **32**, 201–245 (2002)
6. Ben-David, S., Borodin, A., Karp, R.M., Tardos, G., Wigderson, A.: On the power of randomization in on-line algorithms. Algorithmica **11**, 2–14 (1994)

7. Benteley, J.L., McGeoch, C.C.: Amortized analyses of self-organizing sequential search heuristics. Commun. ACM **28**, 404–411 (1985)
8. Bentley, J.L., Sleator, D.S., Tarjan, R.E., Wei, V.K.: A locally adaptive data compression scheme. Commun. ACM **29**, 320–330 (1986)
9. Blum, A., Chawla, S., Kalai, A.: Static optimality and dynamic search-optimality in lists and trees. In: Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1–8 (2002)
10. Burrows, M., Wheeler, D.J.: A block-sorting lossless data compression algorithm. DEC SRC Research Report 124, (1994)
11. Chung, F.R.K., Hajela, D.J., Seymour, P.D.: Self-organizing sequential search and Hilbert's inequality. In: Proc. 17th Annual Symposium on the Theory of Computing pp 217–223 (1985)
12. Irani, S.: Two results on the list update problem. Inf. Proc. Lett. **38**, 301–306 (1991)
13. Karp, R. Raghavan, P.: From a personal communication cited in [14]
14. Reingold, N., Westbrook, J., Sleator, D.D.: Randomized competitive algorithms for the list update problem. Algorithmica **11**, 15–32 (1994)
15. Rivest, R.: On self-organizing sequential search heuristics. Commun. ACM **19**, 63–67 (1976)
16. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**, 202–208 (1985)
17. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM **32**, 652–686 (1985)
18. Witten, I.H., Bell, T.: The Calgary/Canterbury text compression corpus. Anonymous ftp from ftp://ftp.cpsc.ucalgary.ca:/pub/text.compression/corpus/text.compression.corpus.tar.Z

Online Paging and Caching

1985–2002; multiple authors

Neal E. Young
Department of Computer Science,
University of California at Riverside, Riverside, CA, USA

Keywords and Synonyms

Paging; Caching; Weighted caching; Weighted paging; File caching

Problem Definition

A *file-caching* problem instance specifies a cache size k (a positive integer) and a sequence of requests to files, each with a *size* (a positive integer) and a *retrieval cost* (a non-negative number). The goal is to maintain the cache to satisfy the requests while minimizing the retrieval cost. Specifically, for each request, if the file is not in the cache, one must retrieve it into the cache (paying the retrieval cost) and remove other files to bring the total size of files in the cache to k or less. *Weighted caching*, or *weighted paging* is the special case when each file size is 1. *Paging* is the special case when each file size and each retrieval cost is 1. Then the goal is to minimize *cache misses*, or equivalently the *fault rate*.

An algorithm is *online* if its response to each request is independent of later requests. In practice this is generally necessary. Standard worst-case analysis is not meaningful for online algorithms – any algorithm will have some input sequence that forces a retrieval for every request. Yet worst-case analysis can be done meaningfully as follows. An algorithm is $c(h,k)$-*competitive* if on *any* sequence σ the total (expected) retrieval cost incurred by the algorithm using a cache of size k is at most $c(h,k)$ times the *minimum* cost to handle σ with a cache of size h (plus a constant independent of σ). Then the algorithm has *competitive ratio* $c(h,k)$. The study of competitive ratios is called *competitive analysis*. (In the larger context of approximation algorithms for combinatorial optimization, this ratio is commonly called the *approximation ratio*.)

Algorithms

Here are definitions of a number of caching algorithms; first is LandLord. LandLord gives each file "credit" (equal to its cost) when the file is requested and not in cache. When necessary, LandLord reduces all cached file's credits proportionally to file size, then evicts files as they run out of credit.

File-caching algorithm LandLord Maintain real value credit[f] with each file f (credit[f] = 0 if f is not in the cache).

When a file g is requested:

1. **if** g is not in the cache:
2. **until** the cache has room for g:
3. **for each** cached file f: decrease credit[f] by $\Delta \cdot$ size[f],
4. where $\Delta = \min_{f \in \text{cache}} \text{credit}[f]/\text{size}[f]$.
5. Evict from the cache any subset of the zero-credit files f.
6. Retrieve g into the cache; set credit[g] $\leftarrow$ cost(g).
7. **else** Reset credit[g] anywhere between its current value and cost(g).

For weighted caching, file sizes equal 1. Greedy Dual is LandLord for this special case. Balance is the further special case obtained by leaving credit unchanged in line 7.

For paging, files sizes and costs equal 1. Flush-when-full is obtained by evicting *all* zero-credit files in line 5; First-in-first-out is obtained by leaving credits unchanged in line 7 and evicting the file that entered the cache earliest in line 5; Least-recently-used is obtained by raising credits to 1 in line 7 and evicting the least-

recently requested file in line 5. The MARKING algorithm is obtained by raising credits to 1 in line 7 and evicting a *random* zero-credit file in line 5.

Key Results

This entry focuses on competitive analysis of paging and caching strategies as defined above. Competitive analysis has been applied to many problems other than paging and caching, and much is known about other methods of analysis (mainly empirical or average-case) of paging and caching strategies, but these are outside scope of this entry.

Paging

In a seminal paper, Sleator and Tarjan showed that LEAST-RECENTLY-USED, FIRST-IN-FIRST-OUT, and FLUSH-WHEN-FULL are $k/(k-h+1)$-competitive [13]. Sleator and Tarjan also showed that this competitive ratio is the best possible for any deterministic online algorithm.

Fiat et al. showed that the *Marking* algorithm is $2H_k$-competitive and that no randomized online algorithm is better than H_k-competitive [7]. Here $H_k = 1 + 1/2 + \cdots + 1/k \approx .58 + \ln k$. McGeoch and Sleator gave an optimal H_k-competitive randomized online paging algorithm [12].

Weighted caching

For weighted caching, Chrobak et al. showed that the deterministic online BALANCE algorithm is k-competitive [5]. Young showed that GREEDY DUAL is $k/(k-h+1)$-competitive, and that GREEDY DUAL is a primal-dual algorithm – it generates a solution to the linear-programming dual which proves the near-optimality of the primal solution [15]. Recently Bansal et al. used the primal-dual framework to give an $O(\log k)$-competitive randomized algorithm for weighted caching [1].

File caching

When each cost equals 1 (the goal is to minimize the *number* of retrievals), or when each file's cost equals the file's size (the goal is to minimize the total number of *bytes* retrieved), Irani gave $O(\log^2 k)$-competitive randomized online algorithms [8].

For general file caching, Irani and Cao showed that a restriction of LANDLORD is k-competitive [4]. Independently, Young showed that LANDLORD is $k/(k-h+1)$-competitive [15].

Other theoretical models

Practical performance can be better than the worst case studied in competitive analysis. Refinements of the model have been proposed to increase realism. Borodin et al. [3], to model locality of reference, proposed the *access-graph* model (see also [9,10]). Koutsoupias and Papadimitriou proposed the *comparative ratio* (for comparing classes of online algorithms directly) and the *diffuse-adversary model* (where the adversary chooses requests probabilistically subject to restrictions) [11]. Young showed that any $k/(k-h+1)$-competitive algorithm is also *loosely* $O(1)$-competitive: for any fixed $\varepsilon, \delta > 0$, on any sequence, for all but a δ-fraction of cache sizes k, the algorithm either is $O(1)$-competitive or pays at most ε times the sum of the retrieval costs [15].

Analyses of deterministic algorithms

Here is a competitive analysis of GREEDY DUAL for weighted caching.

Theorem 1 GREEDY DUAL *is* $k/(k-h+1)$*-competitive for weighted caching.*

Proof Here is an amortized analysis (in the spirit of Sleator and Tarjan, Chrobak et al., and Young; see [14] for a different primal-dual analysis). Define potential

$$\Phi = (h-1)\cdot \sum_{f\in \text{GD}} \text{credit}[f] + k\cdot \sum_{f\in \text{OPT}} \Big(\text{cost}(f) - \text{credit}[f]\Big),$$

where GD and OPT denote the current caches of GREEDY DUAL and OPT (the optimal off-line algorithm that manages the cache to minimize the total retrieval cost), respectively. After each request, GREEDY DUAL and OPT take (some subset of) the following steps in order.

OPT **evicts a file** f: Since $\text{credit}[f] \le \text{cost}(f)$, Φ cannot increase.

OPT **retrieves requested file** g: OPT pays $\text{cost}(g)$; Φ increases by at most $k\,\text{cost}(g)$.

GREEDY DUAL **decreases** credit$[f]$ **for all** $f \in$ GD: The cache is full and the requested file is in OPT but not yet in GD. So $|\text{GD}| = k$ and $|\text{OPT} \cap \text{GD}| \le h-1$. Thus, the total decrease in Φ is $\Delta[(h-1)|\text{GD}| - k\,|\text{OPT} \cap \text{GD}|] \ge \Delta[(h-1)k - k(h-1)] = 0$.

GREEDY DUAL **evicts a file** f: Since $\text{credit}[f] = 0$, Φ is unchanged.

GREEDY DUAL **retrieves requested file** g **and sets** credit$[g]$ **to** cost(g): GREEDY DUAL pays $c = \text{cost}(g)$. Since g was not in GD but is in OPT, $\text{credit}[g] = 0$ and Φ decreases by $-(h-1)c + k\,c = (k-h+1)c$.

GREEDY DUAL **resets** credit$[g]$ **between its current value and** cost(g): Since $g \in$ OPT and credit$[g]$ only increases, Φ decreases.

So, with each request: (1) when OPT retrieves a file of cost c, Φ increases by at most kc; (2) at no other time does Φ increase; and (3) when GREEDY DUAL retrieves a file of cost c, Φ decreases by at least $(k - h + 1)c$. Since initially $\Phi = 0$ and finally $\Phi \geq 0$, it follows that GREEDY DUAL's total cost times $k - h + 1$ is at most OPT's cost times k.

Extension to file caching

Although the proof above easily extends to LANDLORD, it is more informative to analyze LANDLORD via a *general reduction* from file caching to weighted caching:

Corollary 1 LANDLORD *is* $k/(k - h + 1)$*-competitive for file caching.*

Proof Let W be any deterministic c-competitive weighted-caching algorithm. Define file-caching algorithm F_W as follows. Given request sequence σ, F_W simulates W on weighted-caching sequence σ' as follows. For each file f, break f into size(f) "pieces" $\{f_i\}$ each of size 1 and cost cost(f)/size(f). When f is requested, give a batch $(f_1, f_2, \ldots, f_s)^{N+1}$ of requests for pieces to W. Take N large enough so W has all pieces $\{f_i\}$ cached after the first sN requests of the batch.

Assume that W *respects equivalence*: after each batch, for every file f, all or none of f's pieces are in W's cache. After each batch, make F_W update its cache correspondingly to $\{f : f_i \in \text{cache}(W)\}$. F_W's retrieval cost for σ is at most W's retrieval cost for σ', which is at most $c\,\text{OPT}(\sigma')$, which is at most $c\,\text{OPT}(\sigma')$. Thus, F_W is c-competitive for file caching.

Now, observe that GREEDY DUAL can be made to respect equivalence. When GREEDY DUAL processes a batch of requests $(f_1, f_2, \ldots, f_s)^{N+1}$ resulting in retrievals, for the last s requests, make GREEDY DUAL set credit$[f_i]$ = cost(f_i) = cost$(f)/s$ in line 7. In general, restrict GREEDY DUAL to raise credits of equivalent pieces f_i equally in line 7. After each batch the credits on equivalent pieces f_i will be the same. When GREEDY DUAL evicts a piece f_i, make GREEDY DUAL evict all other equivalent pieces f_j (all will have zero credit).

With these restrictions, GREEDY DUAL respects equivalence. Finally, taking W to be GREEDY DUAL above, F_W is LANDLORD.

Analysis of the randomized MARKING algorithm

Here is a competitive analysis of the MARKING algorithm.

Theorem 2 *The* MARKING *algorithm is* $2H_k$*-competitive for paging.*

Proof Given a paging request sequence σ, partition σ into contiguous *phases* as follows. Each phase starts with the request after the end of the previous phase and continues as long as possible subject to the constraint that it should contain requests to at most k distinct pages. (Each phase starts when the algorithm runs out of zero-credit files and reduces all credits to zero.)

Say a request in the phase is *new* if the item requested was not requested in the previous phase. Let m_i denote the number of new requests in the ith phase. During phases $i-1$ and i, $k + m_i$ distinct files are requested. OPT has at most k of these in cache at the start of the $i-1$st phase, so it will retrieve at least m_i of them before the end of the ith phase. So OPT's total cost is at least $\max\left\{\sum_i m_{2i}, \sum_i m_{2i+1}\right\} \geq \sum_i m_i/2$.

Say a non-new request is *redundant* if it is to a file with credit 1 and non-redundant otherwise. Each new request costs the MARKING algorithm 1. The jth non-redundant request costs the MARKING algorithm at most $m_i/(k - j + 1)$ in expectation because, of the $k - j + 1$ files that if requested would be non-redundant, at most m_i are not in the cache (and each is equally likely to be in the cache). Thus, in expectation MARKING pays at most $m_i + \sum_{j=1}^{k-m_i} m_i/(k - j + 1) \leq m_i H_k$ for the phase, and at most $H_k \sum_i m_i$ total.

Applications

Variants of GREEDY DUAL and LANDLORD have been incorporated into file-caching software such as Squid [6].

Experimental Results

For a study of competitive ratios on practical inputs, see for example [4,6,14].

Cross References

- Algorithm DC-Tree for k Servers on Trees
- Alternative Performance Measures in Online Algorithms
- Online List Update
- Price of Anarchy
- Work-Function Algorithm for k Servers

Recommended Reading

1. Bansal, N., Buchbinder, N., Naor, J.: A primal-dual randomized algorithm for weighted paging. Proceedings of 48th Annual IEEE Symposium on Foundations of Computer Science, pp. 507–517 (2007)
2. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, New York (1998)

3. Borodin, A., Irani, S., Raghavan, P., Schieber B.: Competitive paging with locality of reference. J. Comput. Syst. Sci. **50**(2), 244–258 (1995)
4. Cao, P., Irani, S.: Cost-aware WWW proxy caching algorithms. In: USENIX Symposium on Internet Technologies and Systems, Monterey, December 1997
5. Chrobak, M., Karloff, H., Payne, T.H., Vishwanathan, S.: New results on server problems. SIAM J. Discret. Math. **4**(2), 172–181 (1991)
6. Dilley, J., Arlitt, M., Perret, S.: Enhancement and validation of Squid's cache replacement policy. Hewlett-Packard Laboratories Technical Report HPL-1999–69 (1999)
7. Fiat, A., Karp, R.M., Luby, M., McGeoch, L.A., Sleator, D.D., Young, N.E.: Competitive paging algorithms. J. Algorithms **12**(4), 685–699 (1991)
8. Irani, S.: Page replacement with multi-size pages and applications to Web caching. Algorithmica **33**(3), 384–409 (2002)
9. Irani, S., Karlin, A.R., Phillips, S.: Strongly competitive algorithms for paging with locality of reference. SIAM J. Comput. **25**(3), 477–497 (1996)
10. Karlin, A.R., Phillips, S.J., Raghavan, P.: Markov paging. SIAM J. Comput. **30**(3), 906–922 (2000)
11. Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. SIAM J. Comput. **30**(1), 300–317 (2000)
12. McGeoch, L.A., Sleator, D.D.: A strongly competitive randomized paging algorithm. Algorithmica **6**, 816–825 (1991)
13. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985)
14. Young, N.E.: The *k*-server dual and loose competitiveness for paging. Algorithmica **11**(6), 525–541 (1994)
15. Young, N.E.: On-line file caching. Algorithmica **33**(3), 371–383 (2002)

Online Scheduling

► List Scheduling
► Load Balancing

Optimal Probabilistic Synchronous Byzantine Agreement

1988; Feldman, Micali

Juan Garay
Bell Labs, Murray Hill, NJ, USA

Keywords and Synonyms

Distributed consensus; Byzantine generals problem

Problem Definition

The Byzantine agreement problem (BA) is concerned with multiple processors (parties, "players") all starting with some initial value, agreeing on a common value, despite the possible disruptive or even malicious behavior of some them. BA is a fundamental problem in fault-tolerant distributed computing and secure multi-party computation.

The problem was introduced by Pease, Shostak and Lamport in [18], who showed that the number of faulty processors must be less than a third of the total number of processors for the problem to have a solution. They also presented a protocol matching this bound, which requires a number of communication rounds proportional to the number of faulty processors—exactly $t + 1$, where t is the number of faulty processors. Fischer and Lynch [10] later showed that this number of rounds is necessary in the worst-case run of any deterministic BA protocol. Furthermore, the above assumes that communication takes place in synchronous rounds. Fischer, Lynch and Patterson [11] proved that no completely asynchronous BA protocol can tolerate even a single processor with the simplest form of misbehavior—namely, ceasing to function at an arbitrary point during the execution of the protocol ("crashing").

To circumvent the above-mentioned lower bound on the number of communication rounds and impossibility result, respectively, researchers beginning with Ben-Or [1] and Rabin [19], and followed by many others (e. g., [3,5]) explored the use of randomization. In particular, Rabin showed that linearly resilient BA protocols in expected *constant* rounds were possible, provided that all the parties have access to a "common coin" (i. e., a common source of randomness)—essentially, the value of the coin can be adopted by the non-faulty processors in case disagreement at any given round is detected, a process that is repeated multiple times. This line of research culminated in the unconditional (or information-theoretic) setting with the work of Feldman and Micali [9], who showed an efficient (i. e., polynomial-time) probabilistic BA protocol tolerating the maximal number of faulty processors[1] that runs in expected constant number of rounds. The main achievement of the Feldman–Micali work is to show how to obtain a shared random coin with constant success probability in the presence of the maximum allowed number of misbehaving parties "from scratch".

Randomization has also been applied to BA protocols in the computational (or cryptographic) setting and for weaker failure models. See [6] for an early survey on the subject.

Notations

Consider a set $\mathcal{P} = \{P_1, P_2, \cdots, P_n\}$ of processors (probabilistic polynomial-time Turing machines) out of which t, $t < n$ may not follow the protocol, and even collude and behave in arbitrary ways. These processors are called

[1] Karlin and Yao [14] showed that the maximum number of faulty processors for probabilistic BA is also $t < \frac{n}{3}$, where n is the total number of processors.

faulty; it is useful to model the faulty processors as being coordinated by an adversary, sometimes called a *t-adversary*.

For $1 \leq i \leq n$, let b_i, $b_i \in \{0, 1\}$ denote party P_i's initial value. The work of Feldman and Micali considers the problem of designing a probabilistic BA protocol in the model defined below.

System Model

The processors are assumed to be connected by point-to-point private channels. Such a network is assumed to be synchronous, i. e., the processors have access to a global clock, and thus the computation of all processors can proceed in a lock-step fashion. It is customary to divide the computation of a synchronous network into *rounds*. In each round, processors send messages, receive messages, and perform some local computation.

The t-adversary is computationally unbounded, *adaptive* (i. e., it chooses which processors to corrupt on the fly), and decides on the messages the faulty processors send in a round depending on the messages sent by the non-faulty processors in all previous rounds, including the current round (this is called a *rushing* adversary).

Given the model above, the goal is to solve the problem stated in the ► Byzantine Agreement; that is, for every set of inputs and any behavior of the faulty processors, to have the non-faulty processors output a common value, subject to the additional condition that if they all start the computation with the same initial value, then that should be the output value. The difference with respect to the other entry is that, thanks to randomization, BA protocols here run in expected constant rounds.

Problem 1 (BA)
Input: *Each processor P_i, $1 \leq i \leq n$, has bit b_i.*
Output: *Eventually, each processor P_i, $1 \leq i \leq n$, outputs bit d_i satisfying the following two conditions:*

- Agreement: *For any two non-faulty processors P_i and P_j, $d_i = d_j$.*
- Validity: *If $b_i = b_j = b$ for all non-faulty processors P_i and P_j, then $d_i = b$ for all non-faulty processors P_i.*

In the above definition input and output values are from $\{0, 1\}$. This is without loss of generality, since there is a simple two-round transformation that reduces a multi-valued agreement problem to the binary problem [20].

Key Results

Theorem 1 *Let $t < \frac{n}{3}$. Then there exists a polynomial-time BA protocol running in expected constant number of rounds.*

The number of rounds of the Feldman–Micali BA protocol is expected constant, but there is no bound in the worst case; that is, for every r, the probability that the protocol proceeds for more than r rounds is very small, yet greater than 0—in fact, equal to $2^{-O(r)}$. Further, the non-faulty processors may not terminate in the same round.[2]

The Feldman–Micali BA protocol assumes synchronous rounds. As mentioned above, one of the motivations for the use of randomization was to overcome the impossibility result due to Fischer, Lynch and Paterson [11] of BA in asynchronous networks, where there is no global clock, and the adversary is also allowed to schedule the arrival time of a message sent to a non-faulty processor (of course, faulty processors may not send any message(s)). In [8], Feldman mentions that the Feldman–Micali BA protocol can be modified to work on asynchronous networks, at the expense of tolerating $t < \frac{n}{4}$ faults. In [4], Canetti and Rabin present a probabilistic asynchronous BA protocol for $t < \frac{n}{3}$ that differs from the Feldman–Micali approach in that it is a Las Vegas protocol—i. e., it has non-terminating runs, but when it terminates, it does so in constant expected rounds.

Applications

There exists a one-to-one correspondence, possibility- and impossibility-wise between BA in the unconditional setting as defined above and a formulation of the problem called the "Byzantine generals" by Lamport, Shostak and Pease [16], where there is a distinguished source among the parties sending a value, call it b_s, and the rest of the parties having to agree on it. The Agreement condition remains unchanged; the Validity condition becomes

- Validity: If the source is non-faulty, then $d_i = b_s$ for all non-faulty processors P_i.

A protocol for this version of the problem realizes a functionality called a "broadcast channel" on a network with only point-to-point connectivity. Such a tool is very useful in the context of cryptographic protocols and secure multi-party computation [12]. Probabilistic BA is particularly relevant here, since it provides a constant-round implementation of the functionality. In this respect, without any optimizations, the reported actual number of expected rounds of the Feldman–Micali BA protocol is at most 57.

[2] Indeed, it was shown by Dwork and Moses [7] that at least $t + 1$ rounds are necessary for simultaneous termination. In [13], Goldreich and Petrank combine "the best of both worlds" by showing a BA protocol running in expected constant number of rounds which always terminates within $t + O(\log t)$ rounds.

Recently, Katz and Koo [15] presented a probabilistic BA protocol with an expected number of rounds at most 23.

BA has many other applications. Refer to the ► Byzantine Agreement, as well as to, e. g., [17] for further discussion of other application areas.

Cross References

- ► Asynchronous Consensus Impossibility
- ► Atomic Broadcast
- ► Byzantine Agreement
- ► Randomized Energy Balance Algorithms in Sensor Networks

Recommended Reading

1. Ben-Or, M.: Another advantage of free choice: Completely asynchronous agreement protocols. In: Proc. 22nd Annual ACM Symposium on the Principles of Distributed Computing, 1983, pp. 27–30
2. Ben-Or, M., El-Yaniv, R.: Optimally-resilient interactive consistency in constant time. Distrib. Comput. **16**(4), 249–262 (2003)
3. Bracha, G.: An *O*(log *n*) expected rounds randomized Byzantine generals protocol. J. Assoc. Comput. Mach. **34**(4), 910–920 (1987)
4. Canetti, R., Rabin, T.: Fast asynchronous Byzantine agreement with optimal resilience. In: Proceedings of the 25th Annual ACM Symposium on the Theory of Computing, San Diego, California, 16–18 May 1993, pp. 42–51
5. Chor, B., Coan, B.: A simple and efficient randomized Byzantine agreement algorithm. IEEE Trans. Softw. Eng. **SE-11**(6), 531–539 (1985)
6. Chor, B., Dwork, C.: Randomization in Byzantine Agreement. Adv. Comput. Res. **5**, 443–497 (1989)
7. Dwork, C., Moses, Y.: Knowledge and Common Knowledge in a Byzantine Environment: Crash Failures. Inf. Comput. **88**(2), 156–186 (1990). Preliminary version in TARK'86
8. Feldman, P.: Optimal Algorithms for Byzantine Agreement. Ph. D. thesis, MIT (1988)
9. Feldman, P., Micali, S.: An optimal probabilistic protocol for synchronous Byzantine agreement. SIAM J. Comput. **26**(4), 873–933 (1997). Preliminary version in STOC'88
10. Fischer, M.J., Lynch, N.A.: A Lower Bound for the Time to Assure Interactive Consistency. Inf. Process. Lett. **14**(4), 183–186 (1982)
11. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty processor. J. ACM **32**(2), 374–382 (1985)
12. Goldreich, O.: Foundations of Cryptography, Volumes 1 and 2. Cambridge University Press, Cambridge (2001), (2004)
13. Goldreich, O., Petrank, E.: The Best of Both Worlds: Guaranteeing Termination in Fast Randomized Agreement Protocols. Inf. Process. Lett. **36**(1), 45–49 (1990)
14. Karlin, A., Yao, A.C.: Probabilistic lower bounds for the byzantine generals problem. Unpublished manuscript
15. Katz, J., Koo, C.: On Expected Constant-Round Protocols for Byzantine Agreement. In: Proceedings of Advances in Cryptology–CRYPTO 2006, Santa Barbara, California, August 2006, pp. 445–462. Springer, Berlin Heidelberg New York (2006)
16. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
17. Lynch, N.: Distributed Algorithms, Morgan Kaufmann, San Francisco (1996)
18. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
19. Rabin, M.: Randomized Byzantine Generals. In: Proc. 24th Anual IEEE Symposium on Foundations of Computer Science, 1983, pp. 403–409
20. Turpin, R., Coan, B.A.: Extending binary Byzantine Agreement to multivalued Byzantine Agreement. Inf. Process. Lett. **18**(2), 73–76 (1984)

Optimal Radius

► Distance-Based Phylogeny Reconstruction (Optimal Radius)

Optimal Stable Marriage

1987; Irving, Leather, Gusfield

ROBERT W. IRVING
Department of Computing Science,
University of Glasgow, Glasgow, UK

Keywords and Synonyms

Optimal stable matching

Problem Definition

The classical *Stable Marriage problem* (SM), first studied by Gale and Shapley [5], is introduced in ► Stable Marriage. An instance of SM comprises a set $\mathcal{M} = \{m_1, \ldots, m_n\}$ of n men and a set $\mathcal{W} = \{w_1, \ldots, w_n\}$ of n women, and for each person a *preference list*, which is a total order over the members of the opposite sex. A man's (respectively woman's) preference list indicates his (respectively her) strict order of preference over the women (respectively men). A matching M is a set of n man–woman pairs in which each person appears exactly once. If the pair (m,w) is in the matching M then m and w are *partners* in M, denoted by $w = M(m)$ and $m = M(w)$. Matching M is stable if there is no man m and woman w such that m prefers w to $M(m)$ and w prefers m to $M(w)$.

The key result established in [5] is that at least one stable matching exists for every instance of SM. In general there may be many stable matchings, so the question arises as to what is an appropriate definition for the 'best' stable matching, and how such a matching may be found.

Gale and Shapley described an algorithm to find a stable matching for a given instance of SM. This algorithm

may be applied either from the men's side or from the women's side. In the former case, it finds the so-called *man-optimal* stable matching, in which each man has the best partner, and each woman the worst partner, that is possible in any stable matching. In the latter case, the *woman-optimal* stable matching is found, in which these properties are interchanged. For some instances of SM, the man-optimal and woman-optimal stable matchings coincide, in which case this is the unique stable matching. In general however, there may be many other stable matchings between these two extremes. Knuth [13] was first to show that the number of stable matchings can grow exponentially with n.

Because of the imbalance inherent, in general, in the man-optimal and woman-optimal solutions, several other notions of optimality in SM have been proposed.

A stable matching M is *egalitarian* if the sum

$$\sum_i r(m_i, M(m_i)) + \sum_j r(w_j, M(w_j))$$

is minimized over all stable matchings, where $r(m,w)$ represents the rank, or position, of w in m's preference list, and similarly for $r(w,m)$. An egalitarian stable matching incorporates an optimality criterion that does not overtly favor the members of one sex – though it is easy to construct instances having many stable matchings in which the unique egalitarian stable matching is in fact the man (or woman) optimal.

A stable matching M is *minimum regret* if the value $\max(r(p, M(p))$ is minimized over all stable matchings, where the maximum is taken over all persons p. A minimum regret stable matching involves an optimality criterion based on the least happy member of the society, but again, minimum regret can coincide with man-optimal or woman-optimal in some cases, even when there are many stable matchings.

A stable matching is *rank-maximal* (or *lexicographically maximal*) if, among all stable matchings, the largest number of people have their first choice partner, and subject to that, the largest number have their second choice partner, and so on.

A stable matching M is *sex-equal* if the difference

$$\left| \sum_i r(m_i, M(m_i)) - \sum_j r(w_j, M(w_j)) \right|$$

is minimized over all stable matchings. This definition is an explicit attempt to ensure that one sex is treated no more favorably than the other, subject to the overriding criterion of stability.

In the *weighted* stable marriage problem (WSM), each person has, as before, a strictly ordered preference list, but the entries in this list have associated costs or weights – $wt(m,w)$ represents the weight associated with woman w in the preference list of man m, and likewise for $wt(w,m)$. It is assumed that the weights are strictly increasing along each preference list.

A stable matching M in an instance of WSM is *optimal* if

$$\sum_i wt(m_i, M(m_i)) + \sum_j wt(w_j, M(w_j))$$

is minimized over all stable matchings.

A stable matching M in an instance of WSM is *balanced* if

$$\max\left(\sum_i wt(m_i, M(m_i)), \sum_j wt(w_j, M(w_j)) \right)$$

is minimized over all stable matchings.

These same forms of optimality may be defined in the more general context of the Stable Marriage problem with Incomplete Preference Lists (SMI) –see ► Stable Marriagefor a formal definition of this problem.

Again as described in ► Stable Marriage, the *Stable Roommates* problem (SR) is a non-bipartite generalization of SM, also introduced by Gale and Shapley [5]. In contrast to SM, an instance of SR may or may not admit a stable matching. Irving [9] gave the first polynomial-time algorithm to determine whether an SR instance admits a stable matching, and if so to find one such matching.

There is no concept of man or woman optimal in the SR context, and nor is there any analogue of sex-equal or balanced matching. However, the other forms of optimality introduced above can be defined also for instances of SR and WSR (Weighted Stable Roommates).

A comprehensive treatment of many aspects of the Stable Marriage problem, as of 1989, appears in the monograph of Gusfield and Irving [6].

Key Results

The key to providing efficient algorithms for the various kinds of optimal stable matching is an understanding of the algebraic structure underlying an SM instance, and the discovery of methods to exploit this structure. Knuth [13] attributes to Conway the observation that the set of stable matchings for an SM instance forms a distributive lattice under a natural dominance relation. Irving and Leather [11] characterized this lattice in terms of so-called *rotations* – essentially minimal differences between lattice elements – which can be efficiently computed directly from the preference lists. The rotations form a nat-

ural partial order, the *rotation poset*, and there is a one-to-one correspondence between the stable matchings and the closed subsets of the rotation poset.

Building on these structural results, Gusfield [8] gave a $O(n^2)$ algorithm to find a minimum-regret stable matching, improving an earlier $O(n^4)$ algorithm described by Knuth [13] and attributed to Selkow. Irving et al. [10] showed how application of network flow methods to the rotation poset yield efficient algorithms for egalitarian and rank-maximal stable matchings, as well as for an optimal stable matching in WSM. These algorithms have complexities $O(n^4)$, $O(n^5 \log n \log n)$ and $O(n^4 \log n)$ respectively. Subsequently, by using an interpretation of a stable marriage instance as an instance of 2-SAT, and with the aid of a faster network flow algorithm exploiting the special structure of networks representing SM instances, Feder [3,4] reduced these complexities to $O(n^3)$, $O(n^{3.5})$ and $O(\min(n, \sqrt{K})n^2 \log(K/n^n + 2))$ respectively, where K is the weight of an optimal solution.

By way of contrast, and perhaps surprisingly, the problems of finding a sex-equal stable matching for an instance of SM and of finding a balanced stable matching for an instance of WSM have been shown to be NP-hard [2,12].

The following theorem summarizes the current state of knowledge regarding the various flavors of optimal stable matching in SM and WSM.

Theorem 1 *For an instance of SM:*

(i) *A minimum regret stable matching can be found in $O(n^2)$ time.*

(ii) *An egalitarian stable matching can be found in $O(n^3)$ time.*

(iii) *A rank-maximal stable matching can be found in $O(n^{3.5})$ time.*

(iv) *The problem of finding a sex-equal stable matching is NP-hard.*

For an instance of WSM:

(v) *An optimal stable matching can be found in $O(\min(n, \sqrt{K})n^2 \log(K/n^2 + 2))$ time, where K is the weight of an optimal solution.*

(vi) *The problem of finding a balanced stable matching is NP-hard, but can be approximated within a factor of 2 in $O(n^2)$ time.*

Among related problems that can also be solved efficiently by exploitation of the rotation structure of an instance of SM are the following [8]:

- all stable pairs, i. e., pairs that belong to at least one stable matching, can be found in $O(n^2)$ time;
- all stable matchings can be enumerated in $O(n^2 + kn)$ time, where k is the number of such matchings.

Results analogous to those of Theorem 1 are known for the more general SMI problem. In the case of the Stable Roommates problem (SR), some of these problems appear to be harder, as summarized in the following theorem.

Theorem 2 *For an instance of SR:*

(i) *A minimum regret stable matching can be found in $O(n^2)$ time [7].*

(ii) *The problem of finding an egalitarian stable matching is NP-hard. It can be approximated in polynomial time within a factor of α if and only if minimum vertex cover can be approximated within α [1,2].*

For an instance of WSR (weighted stable roommates):

(iii) *The problem of finding an optimal stable matching is NP-hard, but can be approximated within a factor of 2 in $O(n^2)$ time [3].*

Applications

The best known and most important applications of stable matching algorithms are in centralized matching schemes in the medical and educational domains. ► Hospitals / Residents Problem includes a summary of some of these applications.

Open Problems

There remains the possibility of improving the complexity bounds for some of the optimization problems discussed, and for finding better polynomial-time approximation algorithms for the various NP-hard problems.

Cross References

- ► Hospitals/Residents Problem
- ► Ranked Matching
- ► Stable Marriage and Discrete Convex Analysis
- ► Stable Marriage with Ties and Incomplete Lists
- ► Stable Partition Problem

Recommended Reading

1. Feder, T.: A new fixed point approach for stable networks and stable marriages. In: Proceedings of 21st ACM Symposium on Theory of Computing, pp. 513–522, Theory of Computing, Seattle WA, May 1989, pp. 513–522, ACM, New York (1989)
2. Feder, T.: Stable networks and product graphs. Ph. D. thesis, Stanford University (1991)
3. Feder, T.: A new fixed point approach for stable networks and stable marriages. J. Comput. Syst. Sci. **45**, 233–284 (1992)
4. Feder, T.: Network flow and 2-satisfiability. Algorithmica **11**, 291–319 (1994)
5. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Mon. **69**, 9–15 (1962)

6. Gusfield, D., Irving, R.W.: The Stable Marriage Problem: Structure and Algorithms. MIT Press, Cambrigde MA (1989)
7. Gusfield, D.: The structure of the stable roommate problem: efficient representation and enumeration of all stable assignments. SIAM J. Comput. **17**(4), 742–769 (1988)
8. Gusfield, D.: Three fast algorithms for four problems in stable marriage. SIAM J. Comput. **16**(1), 111–128 (1987)
9. Irving, R.W.: An efficient algorithm for the stable roommates problem. J. Algorithms **6**, 577–595 (1985)
10. Irving, R.W., Leather, P., Gusfield, D.: An efficient algorithm for the "optimal stable" marriage. J. ACM **34**(3), 532–543 (1987)
11. Irving, R.W., Leather, P.: The complexity of counting stable marriages. SIAM J. Comput. **15**(3), 655–667 (1986)
12. Kato, A.: Complexity of the sex-equal stable marriage problem. Jpn. J. Ind. Appl. Math. **10**, 1–19 (1993)
13. Knuth, D.E.: Mariages Stables. Les Presses de L'Université de Montréal (1976)