R

Radiocoloring in Planar Graphs

2005; Fotakis, Nikoletseas, Papadopoulou, Spirakis

VICKY PAPADOPOULOU
Department of Computer Science, University of Cyprus, Nicosia, Cyprus

Keywords and Synonyms

λ-coloring; k-coloring; Distance-2 coloring; Coloring the square of the graph

Problem Definition

Consider a graph $G(V, E)$. For any two vertices $u, v \in V$, $d(u, v)$ denotes the distance of u, v in G. The general problem concerns a coloring of the graph G and it is defined as follows:

Definition 1 (k-coloring problem)
INPUT: A graph $G(V, E)$.
OUTPUT: A function $\phi : V \to \{1, \ldots, \infty\}$, called *$k$-coloring of G* such that $\forall u, v \in V$, $x \in \{0, 1, \ldots, k\}$: if $d(u, v) \geq k - x + 1$ then $|\phi(u) - \phi(v)| = x$.
OBJECTIVE: Let $|\phi(V)| = \lambda_\phi$. Then λ_ϕ is the *number of colors* that φ actually uses (it is usually called *order* of G under φ). The number $\nu_\phi = max_{v \in V}\phi(v) - min_{u \in V}\phi(u) + 1$ is usually called the *span* of G under φ. The function φ satisfies one of the following objectives:

- minimum span: λ_ϕ is the minimum possible over all possible functions φ of G;
- minimum order: ν_ϕ is the minimum possible over all possible functions φ of G;
- Min span order: obtains a minimum span and moreover, from all minimum span assignments, φ obtains a minimum order.
- Min order span: obtains a minimum order and moreover, from all minimum order assignments, φ obtains a minimum span.

Note that the case $k = 1$ corresponds to the well known problem of *vertex graph coloring*. Thus, k-coloring problem (with k as an input) is $\mathcal{NP}$-complete [4]. The case of k-coloring problem where $k = 2$, is called the *Radiocoloring problem*.

Definition 2 (Radiocoloring Problem (RCP) [7])
INPUT: A graph $G(V, E)$.
OUTPUT: A function $\Phi : V \to N^*$ such that $|\Phi(u) - \Phi(v)| \geq 2$ if $d(u, v) = 1$ and $|\Phi(u) - \Phi(v)| \geq 1$ if $d(u, v) = 2$.
OBJECTIVE: The least possible number (order) needed to radiocolor G is denoted by $X_{\text{order}}(G)$. The least possible number $\max_{v \in V} \Phi(v) - \min_{u \in V} \Phi(u) + 1$ (span) needed for the radiocoloring of G is denoted as $X_{\text{span}}(G)$. Function Φ satisfies one of the followings:

- Min span RCP: Φ obtains a minimum span, i. e. $\lambda_\Phi = X_{span}(G)$;
- Min order RCP: Φ obtains a minimum order $\nu_\Phi = X_{\text{order}}(G)$;
- Min span order RCP: obtains a minimum span and moreover, from all minimum span assignments, Φ obtains a minimum order.
- Min order span RCP: obtains a minimum order and moreover, from all minimum order assignments, Φ obtains a minimum span.

A related to the RCP problem concerns to the square of a graph G, which is defined as follows:

Definition 3 Given a graph $G(V, E)$, G^2 is the graph having the same vertex set V and an edge set $E' : \{u, v\} \in E'$ iff $d(u, v) \leq 2$ in G.

The related problem is to color the square of a graph G, G^2 so that no two neighbor vertices (in G^2) get the same color. The objective is to use a minimum number of colors, denoted as $\chi(G^2)$ and called *chromatic number of the square of the graph G*. [5,6] first observed that for any graph G, $X_{\text{order}}(G)$ is the same as the (vertex) chromatic number of G^2, i. e. $X_{\text{order}}(G) = \chi(G^2)$.

Key Results

[5,6] studied *min span order*, *min order* and *min span* RCP in *planar* graph G. A planar graph, is a graph for which its edges can be embedded in the plane without crossings. The following results are obtained:

- It is first shown that the number of colors used in the *min span order RCP* of graph G is different from the chromatic number of the square of the graph, $\chi(G^2)$. In particular, it may be greater than $\chi(G^2)$.
- It is then proved that the radiocoloring problem for general graphs is hard to approximate (unless $\mathcal{NP} = ZPP$, the class of problems with polynomial time zero-error randomized algorithms) within a factor of $n^{1/2-\epsilon}$ (for any $\epsilon > 0$), where n is the number of vertices of the graph. However, when restricted to some special cases of graphs, the problem becomes easier.
  It is shown that the *min span RCP* and *min span order RCP* are *$\mathcal{NP}$-complete* for planar graphs. Note that few combinatorial problems remain hard for *planar* graphs and their proofs of hardness are not easy since they have to use planar gadgets which are difficult to find and understand.
- It presents a $O(n\Delta(G))$ time algorithm that *approximates* the min order of RCP, X_{order}, of a planar graph G *by a constant ratio which tends to 2* as the maximum degree $\Delta(G)$ of G increases.
  The algorithm presented is motivated by a constructive coloring theorem of Heuvel and McGuiness [9]. The construction of [9] can lead (as shown) to an $O(n^2)$ technique assuming that a planar embedding of G is given. [5,6] improves the time complexity of the approximation, and presents a much more simple algorithm to verify and implement. The algorithm does not need any planar embedding as input.
- Finally, the work considers the problem of *estimating the number of different radiocolorings* of a planar graph G. This is a $\#\mathcal{P}$-complete problem (as can be easily seen from the completeness reduction presented there that can be done parsimonious). They authors employ here standard techniques of rapidly mixing Markov Chains and the *new method of coupling* for purposes of proving *rapid convergence* (see e. g. [10]) and present *a fully polynomial randomized approximation scheme* for estimating the number of radiocolorings with λ colors for a planar graph G, when $\lambda \geq 4\Delta(G) + 50$.

In [8] and [7] it has been proved that the problem of min span RCP is $\mathcal{NP}$-complete, even for graphs of diameter 2. The reductions use highly non-planar graphs. In [11] it is proved that the problem of coloring the square of a general graph is $\mathcal{NP}$-complete.

Another variation of RCP for planar graphs, called *distance-2-coloring* is studied in [12]. This is the problem of coloring a given graph G with the minimum number of colors so that the vertices of distance *at most* two get different colors. Note that this problem is equivalent to coloring the square of the graph G, G^2. In [12] it is proved that the distance-2-coloring problem for planar graphs is $\mathcal{NP}$-complete. As it is shown in [5,6], this problem is different from the min span order RCP. Thus, the $\mathcal{NP}$-completeness proof in [12] certainly does not imply the $\mathcal{NP}$-completeness of min span order RCP proved in [5,6]. In [12] a 9-approximation algorithm for the distance-2-coloring of planar graphs is also provided.

Independently and in parallel, Agnarsson and Halldórsson in [1] presented approximations for the chromatic number of square and power graphs (G^k). In particular they presented an 1.8-approximation algorithm for coloring the square of a planar graph of large degree ($\Delta(G) \geq 749$). Their method utilizes the notion of *inductiveness* of the square of a planar graph.

Bodlaender et al. in [2] proved also independently and and in parallel that the min span RCP, called λ-labeling there, is $\mathcal{NP}$-complete for planar graphs, using a similar to the approach used in [5,6]. In the same work the authors presented approximations for the problem for some interesting families of graphs: outerplanar graphs, graphs of bounded treewidth, permutation and split graphs.

Applications

The Frequency Assignment Problem (FAP) in radio networks is a well-studied, interesting problem, aiming at assigning frequencies to transmitters exploiting frequency reuse while keeping signal interference to acceptable levels. The interference between transmitters are modeled by an interference graph $G(V, E)$, where V ($|V| = n$) corresponds to the set of transmitters and E represents distance constraints (e. g. if two neighbor nodes in G get the same or close frequencies then this causes unacceptable levels of interference). In most real life cases the network topology formed has some special properties, e. g. G is a lattice network or a planar graph. Planar graphs are mainly the object of study in [5,6].

The FAP is usually modeled by variations of the graph coloring problem. The set of colors represents the available frequencies. In addition, each color in a particular assignment gets an integer value which has to satisfy certain inequalities compared to the values of colors of nearby nodes in G (frequency-distance constraints). A discrete version of FAP is the k-coloring problem, of which a particular instance, for $k = 2$, is investigated in [5,6].

Real networks reserve bandwidth (range of frequencies) rather than distinct frequencies. In this case, an assignment seeks to use as small range of frequencies as possible. It is sometimes desirable to use as few distinct frequencies of a given bandwidth (span) as possible, since the unused frequencies are available for other use. However, there are cases where the primary objective is to minimize the number of frequencies used and the span is a secondary objective, since we wish to avoid reserving unnecessary large span. These realistic scenaria directed researchers to consider optimization versions of the RCP, where one aims in minimizing the span (bandwidth) or the order (distinct frequencies used) of the assignment. Such optimization problems are investigated in [5,6].

Cross References

- ► Channel Assignment and Routing in Multi-Radio Wireless Mesh Networks
- ► Graph Coloring

Recommended Reading

1. Agnarsson, G., Halldórsson, M.M.: Coloring Powers of Planar Graphs. In: Proceedings of the 11th Annual ACM-SIAM symposium on Discrete algorithms, pp. 654–662 (2000)
2. Bodlaender, H.L., Kloks, T., Tan, R.B., van Leeuwen, J.: Approximations for λ-Coloring of Graphs. In: Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science, vol. 1770, pp. 395-406. Springer (2000)
3. Hale, W.K.: Frequency Assignment: Theory and Applications. In: Proceedings of the IEEE, vol. 68, number 12, pp. 1497-1514 (1980)
4. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness, W.H. Freeman and Co. (1979)
5. Fotakis, D., Nikoletseas, S., Papadopoulou, V., Spirakis, P.: $\mathcal{NP}$-Completeness Results and Efficient Approximations for Radiocoloring in Planar Graphs. In: Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes of Computer Science, vol. 1893, pp. 363–372. Springer (2000)
6. Fotakis, D., Nikoletseas, S., Papadopoulou, V.G., Spirakis, P.G.: Radiocoloring in Planar Graphs: Complexity and Approximations. Theor. Comput. Sci. Elsevier **340**, 514–538 (2005)
7. Fotakis, D., Pantziou, G., Pentaris, G., Spirakis, P.: Frequency Assignment in Mobile and Radio Networks. In: Networks in Distributed Computing, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 45, pp. 73–90 (1999)
8. Griggs, J., Liu, D.: Minimum Span Channel Assignments. In: Recent Advances in Radio Channel Assignments, Invited Minisymposium, Discrete Mathematics (1998)
9. van d. Heuvel, J., McGuiness, S.: Colouring the Square of a Planar Graph. CDAM Research Report Series, July 1999
10. Jerrum, M.: A very simple Algorithm for Estimating the Number of k-colourings of a Low Degree Graph. Random Struct. Algorithms **7**, 157–165 (1994)
11. Lin, Y.L., Skiena, S.: Algorithms for Square Roots of Graphs. SIAM J. Discret. Math. **8**, 99–118 (1995)
12. Ramanathan, S., Loyd, E.R.: The Complexity of Distance 2-Coloring. In: Proceedings of the 4th International Conference of Computing and Information, pp. 71–74 (1992)

Randomization in Distributed Computing

1996; Chandra

Tushar Deepak Chandra
IBM Watson Research Center, Yorktown Heights, NY, USA

Keywords and Synonyms

Agreement; Byzantine agreement

Problem Definition

This problem is concerned with using the multi-writer multi-reader register primitive in the shared memory model to design a fast, wait-free implementation of consensus. Below are detailed descriptions of each of these terms.

Consensus Problems

There are n processors and the goal is to design distributed algorithms to solve the following two consensus problems for these processors.

Problem 1 (Binary consensus)
Input: Processor i has input bit b_i.
Output: *Each processor i has output bit b'_i such that: 1) all the output bits b'_i equal the same value v; and 2) $v = b_i$ for some processor i.*

Problem 2 (Id consensus)
Input: *Processor i has a unique id u_i.*
Output: *Each processor i has output value u'_i such that: 1) all the output values u'_i equal the same value u; and 2) $u = u_i$ for some processor i.*

Wait-Free

This result builds on extensive previous work on the shared memory model of parallel computing. Shared object types include data structures such as read/write registers and synchronization primitives such as "test and set".

A shared object is said to be *wait-free* if it ensures that every invocation on the object is guaranteed a response in finite time even if some or all of the other processors in the system crash. In this problem, the existence of wait-free registers is assumed and the goal is to create a fast wait-free algorithm to solve the consensus problem. In the rest of this summary, "wait-free implementations" will be referred to simply as "implementations" i. e. the term wait-free will be omitted.

Multi-writer Multi-reader Register

Many past results on solving consensus in the shared memory model assume the existence of a single writer multi-reader register. For such a register, there is a single writer client and multiple reader clients. Unfortunately, it is easy to show that the per processor step complexity of any implementation of consensus from single writer multi-reader registers will be at least linear in the number of processors. Thus, to achieve a time efficient implementation of consensus, the more powerful primitive of a multi-writer multi-reader register must be assumed. A multi-writer multi-reader register assumes the clients of the register are multiple writers and multiple readers. It is well known that it is possible to implement such a register in the shared memory model.

The Adversary

Solving the above problems is complicated by the fact that the programmer has little control over the rate at which individual processors execute. To model this fact, it is assumed that the schedule at which processors run is picked by an adversary. It is well-known that there is no deterministic algorithm that can solve either Binary consensus or ID consensus in this adversarial model if the number of processors is greater than 1 [6,7]. Thus, researchers have turned to the use of randomized algorithms to solve this problem [1]. These algorithms have access to random coin flips. Three types of adversaries are considered for randomized algorithms. The *strong adversary* is assumed to know the outcome of a coin flip immediately after the coin is flipped and to be able to modify its schedule accordingly. The *oblivious adversary* has to fix the schedule before any of the coins are flipped. The *intermediate adversary* is not permitted to see the outcome of a coin flip until some process makes a choice based on that coin flip. In particular, a process can flip a coin and write the result in a global register, but the intermediate adversary does not know the outcome of the coin flip until some process reads the value written in the register.

Key Results

Theorem 1 *Assuming the existence of multi-writer multi-reader registers, there exists a randomized algorithm to solve binary consensus against an intermediate adversary with $O(1)$ expected steps per processor.*

Theorem 2 *Assuming the existence of multi-writer multi-reader registers, there exists a randomized algorithm to solve id-consensus against an intermediate adversary with $O(\log^2 n)$ expected steps per processor.*

Both of these results assume that every processor has a unique identifier. Prior to this result, the fastest known randomized algorithm for binary consensus made use of single writer multiple reader registers, was robust against a strong adversary, and required $O(n \log^2 n)$ steps per processor [2]. Thus, the above improvements are obtained at the cost of weakening the adversary and strengthening the system model when compared to [2].

Applications

Binary consensus is one of the most fundamental problems in distributed computing. An example of its importance is the following result shown by Herlihy [8]: If an abstract data type X together with shared memory is powerful enough to implement wait-free consensus, then X together with shared memory is powerful enough to implement in a wait-free manner any other data structure Y. Thus, using this result, a wait-free version of any data structure can be created using only wait-free multi-writer multi-reader registers as a building block.

Binary consensus has practical applications in many areas including: database management, multiprocessor computation, fault diagnosis, and mission-critical systems such as flight control. Lynch contains an extensive discussion of some of these application areas [9].

Open Problems

This result leaves open several problems. First, it leaves open a gap on the number of steps per process required to perform randomized consensus using multi-writer multi-reader registers against the *strong* adversary. A recent result by Attiya and Censor shows an $\Omega(n^2)$ lower bound on the total number of steps for all processors with multi-writer multi-reader registers (implying $\Omega(n)$ steps per process) [3]. They also show a matching upper bound of $O(n^2)$ on the total number of steps. However, closing the gap on the per-process number of steps is still open.

Another open problem is whether there is a randomized implementation of id consensus using multi-reader

multi-writer registers that is robust to the intermediate adversary and whose expected number of steps per processor is better than $O(\log^2 n)$. In particular, is a constant run time possible? Aumann in follow up work to this result was able to improve the expected run time per process to $O(\log n)$ [4]. However, to the best of the reviewer's knowledge, there have been no further improvements.

A third open problem is to close the gap on the time required to solve binary consensus against the strong adversary with a single writer multiple reader register. The fastest known randomized algorithm in this scenario requires $O(n \log^2 n)$ steps per processor [2]. A trivial lower bound on the number of steps per processor when single-writer registers are used is $\Omega(n)$. However, to the best of this reviewers knowledge, a $O(\log^2 n)$ gap still remains open.

A final open problem is to close the gap on the total work required to solve consensus with single-reader single-writer registers against an oblivious adversary. Aumann and Kapah-Levy describe algorithms for this scenario that require $O(n \log n \exp(2\sqrt{\ln n \ln(c \log n \log^* n)}$ expected total work for some constant c [5]. In particular, the total work is less than $O(n^{1+\epsilon})$ for any $\epsilon > 0$. A trivial lower bound on total work is $\Omega(n)$, but a gap remains open.

Cross References

- ▶ Asynchronous Consensus Impossibility
- ▶ Atomic Broadcast
- ▶ Byzantine Agreement
- ▶ Implementing Shared Registers in Asynchronous Message-Passing Systems
- ▶ Optimal Probabilistic Synchronous Byzantine Agreement
- ▶ Registers
- ▶ Set Agreement
- ▶ Snapshots in Shared Memory
- ▶ Wait-Free Synchronization

Recommended Reading

1. Aspnes, J.: Randomized protocols for asynchronous consensus. Distrib. Comput. **16**(2–3), 165–175 (2003)
2. Aspnes, J., Waarts, O.: Randomized consensus in expected $o(n \log^2 n)$ operations per processor. In: Proceedings of the 33rd Symposium on Foundations of Computer Science. 24–26 October 1992, pp. 137–146. IEEE Computer Society, Pittsburgh (1992)
3. Attiya, H., Censor, K.: Tight bounds for asynchronous randomized consensus. In: Proceedings of the Symposium on the Theory of Computation. San Diego, 11–13 June 2007 ACM Special Interest Group on Algorithms and Computation Theory (SIGACT) (2007)
4. Aumann, Y.: Efficient asynchronous consensus with the weak adversary scheduler. In: Symposium on Principles of Distrib. Comput.(PODC) Santa Barbara, 21–24 August 1997, pp. 209–218. ACM Special Interest Group on Algorithms and Computation Theory (SIGACT) (1997)
5. Aumann, Y., Kapach-Levy, A.: Cooperative sharing and asynchronous consensus using single-reader/single-writer registers. In: Proceedings of 10th Annual ACM-SIAM Symposium of Discrete Algorithms (SODA) Baltimore, 17–19 January 1999, pp. 61–70. Society for Industrial and Applied Mathematics (SIAM) (1999)
6. Dolev, D., Dwork, C., Stockmeyer, L.: On the minimal synchronism needed for distributed consensus. J. ACM (JACM) **34**(1), 77–97 (1987)
7. Fischer, M.J., Lynch, N.A., Paterson, M.: Impossibility of distributed consensus with one faulty process. In: Proceedings of the 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database System (PODS) Atlante, 21–23 March, pp. 1–7. Association for Computational Machinery (ACM) (1983)
8. Herlihy, M.: Wait-free synchronization. ACM Trans. Programm. Lang. Syst. **13**(1), 124–149 (1991)
9. Lynch, N.: Distributed Algorithms. Morgan Kaufmann, San Mateo (1996)

Randomized Broadcasting in Radio Networks

1992; Reuven Bar-Yehuda, Goldreich, Itai

Alon Itai
Depart of Computer Science, Technion, Haifa, Israel

Keywords and Synonyms

Multi-hop radio networks; Ad hoc networks

Problem Definition

The paper investigates deterministic and randomized protocols for achieving broadcast (distributing a message from a source to all other nodes) in arbitrary multi-hop synchronous radio networks.

The model consists of an arbitrary (undirected) network, with processors communicating in synchronous time-slots subject to the following rules. In each time-slot, each processor acts either as a *transmitter* or as a *receiver*. A processor acting as a receiver is said to receive a message in time-slot t if exactly one of its neighbors transmits in that time-slot. The message received is the one transmitted. If more than one neighbor transmits in that time-slot, a *conflict* occurs. In this case the receiver may either get a message from one of the transmitting neighbors or get no message. It is assumed that conflicts (or "collisions") are not detected, hence a processor cannot distinguish the case in which no neighbor transmits from the case in which two

or more of its neighbors transmits during that time-slot. The processors are not required to have ID's nor do they know their neighbors, in particular the processors do not know the topology of the network.

The only inputs required by the protocol are the number of processors in the network – n, Δ – an a priori known upper bound on the maximum degree in the network and the error bound –ϵ. (All bounds are a priori known to the algorithm.)

Broadcast is a task initiated by a single processor, called the *source*, transmitting a single *message*. The goal is to have the message reach all processors in the network.

Key Results

The main result is a randomized protocol that achieves broadcast in time which is optimal up to a logarithmic factor. In particular, with probability $1-\epsilon$, the protocol achieves broadcast within $O((D+\log n/\epsilon)\cdot\log n)$ time-slots.

On the other hand, a linear lower bound on the deterministic time-complexity of broadcast is proved. Namely, any deterministic broadcast protocol requires $\Omega(n)$ time-slots, even if the network has diameter 3, and n is known to all processors. These two results demonstrate an exponential gap in complexity between randomization and determinism.

Randomized Protocols

The Procedure *Decay* The basic idea used in the protocol is to resolve potential conflicts by randomly eliminating half of the transmitters. This process of "cutting by half" is repeated each time-slot with the hope that there will exist a time-slot with a single active transmitter. The "cutting by half" process is easily implemented distributively by letting each processor decide randomly whether to eliminate itself. It will be shown that if all neighbors of a receiver follow the elimination procedure then with positive probability there exists a time slot in which exactly one neighbor transmits.

What follows is a description of the procedure for sending a message m, that is executed by each processor after receiving m:

procedure $Decay(k,m)$;
  repeat at most k times (but at least once!)
    send m to all neighbors;
    set $coin \leftarrow 0$ or 1 with equal probability.
  until $coin = 0$.

By using elementary probabilistic arguments, one can prove:

Theorem 1 *Let y be a vertex of G. Also let $d \geq 2$ neighbors of y execute Decay during the time interval $[0,k)$ and assume that they all start the execution at $Time = 0$. Then $P(k,d)$, the probability that y receives a message by $Time = k$, satisfies:*
1. $\lim_{k\to\infty} P(k,d) \geq \frac{2}{3}$;
2. for $k \geq 2\lceil\log d\rceil$, $P(k,d) > \frac{1}{2}$.
(All logarithms are to base 2.)

The expected termination time of the algorithm depends on the probability that $coin = 0$. Here, this probability is set to be one half. An analysis of the merits of using other probabilities was carried out by Hofri [4].

The Broadcast Protocol The broadcast protocol makes several calls to $Decay(k,m)$. By Theorem 1 (2), to ensure that the probability of a processor y receiving the message be at least 1/2, the parameter k should be at least $2\log d$ (where d is the number of neighbors sending a message to y). Since d is not known, the parameter was chosen as $k = 2\lceil\log\Delta\rceil$ (recall that Δ was defined to be an upper bound on the in-degree). Theorem 1 also requires that all participants start executing *Decay* at the same time-slot. Therefore, *Decay* is initiated only at integer multiples of $2\lceil\log\Delta\rceil$.

procedure *Broadcast*;
  $k = 2\lceil\log\Delta\rceil$;
  $t = 2\lceil\log(N/\epsilon)\rceil$;
  Wait until receiving a message, say m;
  do t times {
    Wait until $(Time$ **mod** $k) = 0$;
    $Decay(k,m)$;
  }

A network is said to execute the *Broadcast_scheme* if some processor, denoted s, transmits an initial message and each processor executes the above *Broadcast* procedure.

Theorem 2 *Let $T = 2D + 5\max\{\sqrt{D}, \sqrt{\log(n/\epsilon)}\}\cdot\sqrt{\log(n/\epsilon)}$. Assume that Broadcast_scheme starts at $Time = 0$. Then, with probability $\geq 1-2\epsilon$, by time $2\lceil\log\Delta\rceil\cdot T$ all nodes will receive the message. Furthermore, with probability $\geq 1-2\epsilon$, all the nodes will terminate by time $2\lceil\log\Delta\rceil\cdot(T+\lceil\log(N/\epsilon)\rceil)$.*

The bound provided by Theorem 2 contains two additive terms: the first represents the diameter of the network and the second represents delays caused by conflicts (which are rare, yet they exist).

Additional Properties of the Broadcast Protocol:

- **Processor IDs** – The protocol does not use processor IDs, and thus does not require that the processors have

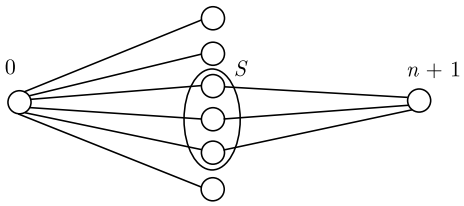
distinct IDs (or that they know the identity of their neighbors). Furthermore, a processor is not even required to know the number of its neighbors. This property makes the protocol adaptive to changes in topology which occur throughout the execution, and resilient to non-malicious faults.

- **Knowing the size of the network** – The protocol performs almost as well when given instead of the actual number of processors (i. e., n), a "good" upper bound on this number (denoted N). An upper bound polynomial in n yields the same time-complexity, up to a constant factor (since complexity is logarithmic in N).
- **Conflict detection** – The algorithm and its complexity remain valid even if no messages can be received when a conflict occurs.
- **Simplicity and fast local computation** – In each time slot each processor performs a constant amount of local computation.
- **Message complexity** – Each processor is active for $\lceil \log(N/\epsilon) \rceil$ consecutive phases and the average number of transmissions per phase is at most 2. Thus the expected number of transmissions of the entire network is bounded by $2n \cdot \lceil \log(N/\epsilon) \rceil$.
- **Adaptiveness to changing topology and fault resilience** – The protocol is resilient to some changes in the topology of the network. For example, edges may be added or deleted at any time, provided that the network of unchanged edges remains connected. This corresponds to fail/stop failure of edges, thus demonstrating the resilience to some non-malicious failures.
- **Directed networks** – The protocol does not use acknowledgments. Thus it may be applied even when the communication links are not symmetric, i. e., the fact that processor v can transmit to u does not imply that u can transmit to v. (The appropriate network model is, therefore, a directed graph.) In real life this situation occurs, for instance, when v has a stronger transmitter than u.

A Lower Bound on Deterministic Algorithms

For deterministic algorithms one can show a lower bound: for every n there exist a family of n-node networks such that every deterministic broadcast scheme requires $\Omega(n)$ time. For every **non-empty** subset $S \subseteq \{1, 2, \ldots, n\}$, consider the following network G_S (Fig. 1).

Node 0 is the *source* and node $n+1$ the *sink*. The source initiates the message and the problem of broadcast in G_S is to reach the sink. The difficulty stems from the fact that the partition of the middle layer (i. e., S) is not known a priori. The following theorem can be proved by a series



Randomized Broadcasting in Radio Networks, Figure 1
The network used for the lower bound

of reductions to a certain "hitting game":

Theorem 3 *Every deterministic broadcast protocol that is correct for all n-node networks requires time* $\Omega(n)$.

In [2] there was some confusion concerning the broadcast model. In that paper it was erroneously claimed that the lower bound holds also when a collision is indistinguishable from the absence of transmission. Kowalski and Pelc [5] disproved this claim by showing how to broadcast in logarithmic time on all networks of type G_S.

Applications

The procedure *Decay* has been used to resolve contention in radio and cellular phone networks.

Cross Reference

- ▶ Broadcasting in Geometric Radio Networks
- ▶ Communication in Ad Hoc Mobile Networks Using Random Walks
- ▶ Deterministic Broadcasting in Radio Networks
- ▶ Randomized Gossiping in Radio Networks

Recommended Reading

Subsequent papers showed the optimality of the randomized algorithm:

- Alon et al. [1] showed the existence of a family of radius-2 networks on n vertices for which any broadcast schedule requires at least $\Omega(\log^2 n)$ time slots.
- Kushilevitz and Mansour [7] showed that for any randomized broadcast protocol there exists a network in which the expected time to broadcast a message is $\Omega(D \log(N/D))$.
- Bruschi and Del Pinto [3] showed that for any deterministic distributed broadcast algorithm, any n and $D \leq n/2$ there exists a network with n nodes and diameter D such that the time needed for broadcast is $\Omega(D \log n)$.
- Kowalski and Pelc [6] discussed networks in which collisions are indistinguishable from the absence of trans-

mission. They showed an $\Omega(n \log n / \log(n/D))$ lower bound and an $O(n \log n)$ upper bound. For this model, they also showed an $O(D \log n + \log^2 n)$ randomized algorithm, thus matching the lower bound of [1] and improving the bound of [2] for graphs for which $D = \theta(n/\log n)$.

1. Alon, N., Bar-Noy, A., Linial, N., Peleg, D.: A lower bound for radio broadcast. J. Comput. Syst. Sci. **43**(2), 290–298 (1991)
2. Bar-Yehuda, R., Goldreich, O., Itai, A.: On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. J. Comput. Syst. Sci. **45**(1), 104–126 (1992)
3. Bruschi, D., Del Pinto, M.: Lower bounds for the broadcast problem in mobile radio networks. Distrib. Comput. **10**(3), 129–135 (1997)
4. Hofri, M.: A feedback-less distributed broadcast algorithm for multihop radio networks with time-varying structure. In: Computer Performance and Reliability, pp. 353–368. (1987)
5. Kowalski, D.R., Pelc, A.: Deterministic broadcasting time in radio networks of unknown topology. In: FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science, Washington, DC, USA, pp. 63–72. IEEE Computer Society (2002)
6. Kowalski, D.R., Pelc, A.: Broadcasting in undirected ad hoc radio networks. Distrib. Comput. **18**(1), 43–57 (2005)
7. Kushilevitz, E., Mansour, Y.: An $\Omega(d \log(n/d))$ lower bound for broadcast in radio networks. In: PODC, 1993, pp. 65–74

Randomized Energy Balance Algorithms in Sensor Networks

2005; Leone, Nikoletseas, Rolim

Pierre Leone[1], Sotiris Nikoletseas[2], José Rolim[1]
[1] Informatics Department, University of Geneva, Geneva, Switzerland
[2] Computer Engineering and Informatics, Department and CTI, University of Patras, Patras, Greece

Keywords and Synonyms

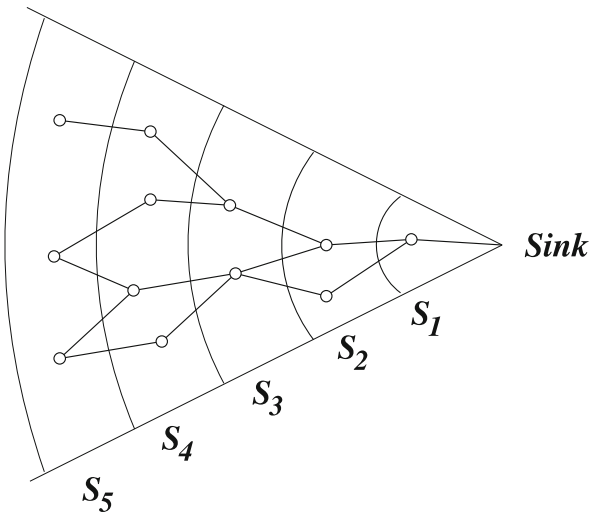
Power conservation

Problem Definition

Recent developments in wireless communications and digital electronics have led to the development of extremely small in size, low-power, low-cost sensor devices (often called smart dust). Such tiny devices integrate sensing, data processing and wireless communication capabilities. Examining each such resource constraint device individually might appear to have small utility; however, the distributed self-collaboration of large numbers of such devices into an ad hoc network may lead to the efficient accomplishment of large sensing tasks i. e., reporting data about the realization of a local event happening in the network area to a faraway control center.

The problem considered is the development of a randomized algorithm to balance energy among sensors whose aim is to detect events in the network area and report them to a sink. The network is sliced by the algorithm into layers composed of sensors at approximately equal distances from the sink [1,2,8] (Fig. 1). The slicing of the network depends on the communication distance. The sink initiates the process by sending a control message containing a counter, the value of which is initially 1. Sensors receiving the message assign themselves to a slice number corresponding to the counter, increment the counter and propagate the message in the network. A sensor already assigned to a slice ignores subsequent received control messages.

The strategy suggested to balance the energy among sensors consists in allowing a sensor to probabilistically choose between either sending data to a sensor in the next layer towards the sink or sending the data directly to the sink. The difference between the two choices is the energy consumption, which is much higher if the sensor decides to report to the sink directly. The energy consumption is modeled as a function of the transmission distance by assuming that the energy necessary to send data up to a distance d is proportional to d^2. Actually, more accurate models can be considered, in which the dependence is of the form d^α, with $2 \leq \alpha \leq 5$ depending on the particular environmental conditions. Although the model chosen



Randomized Energy Balance Algorithms in Sensor Networks, Figure 1
The sink and five slices $S_1, \ldots, S_5$

determines the parameters of the algorithm, the particular shape of the function describing the relationship between the distance of transmission and energy consumption is not relevant except that it might increase with distance. The distance between two successive slices is normalized to be 1. Hence, a sensor sending data to one of its neighbors consumes one unit of energy and a sensor located in slice i consumes i^2 units of energy to report to the sink directly. Small hop transmissions are cheap (with respect to energy consumption) but pass through the critical region around the sink and might strain sensors in that region, while expensive direct transmissions bypass that critical area.

Energy balance is defined as follows:

Definition 1 The network is energy-balanced if the average per sensor energy dissipation is the same for all sectors, i. e., when

$$\frac{E[\mathcal{E}_i]}{S_i} = \frac{E[\mathcal{E}_j]}{S_j}, \quad i, j = 1, \ldots, n \tag{1}$$

where $\mathcal{E}_i$ is the total energy available and S_i is the number of nodes in slice number i.

The dynamics of the network is modeled by assigning probabilities λ_i, $i = 1, \ldots, N$, $\sum \lambda_i = 1$, of the occurrence of an event in slice i. The protocol consists in transmitting the data to a neighbor slice with probability p_i and with probability $1 - p_i$ to the sink, for a sensor belonging to slice i. Hence, the mean energy consumption per data unit is $p_i + (1 - p_i)i^2$. A central assumption in the following is that the events are evenly generated in a given slice. Then, denoting by e_i the energy available per node in slice i (i. e., $e_i = \mathcal{E}_i/S_i$), the problem of energy-balanced data propagation can be formally stated as follows:

Given $\lambda_i, e_i, S_i, i = 1, \ldots, N$, find p_i, λ such that

$$\underbrace{\left(\lambda_i + \lambda_{i+1} p_{i+1} + \ldots + \lambda_n p_n p_{n-1} \cdots p_{i+1}\right)}_{=:x_i} \cdot \left(p_i \frac{1}{S_i} + (1 - p_i)\frac{i^2}{S_i}\right) = \lambda e_i, \quad i = 1, \ldots, N. \tag{2}$$

Equation (2) amounts to ensuring that the mean energy dissipation for all sensors is proportional to the available energy. In turn, this ensures that sensors might, on average, run out of energy all at the same time. Notice that (2) contains the definitions of the x_i. They are the ones estimated in the pseudo-code in Fig. 2, the successive estimations being denoted as $\tilde{x}_i$. These variables are proportional to the number of messages handled by slice i.

```
Initialize x̃_0 = λ, ..., x̃_n
Initialize NbrLoop=1
repeat forever
   Send x̃_i and λ values to the stations which compute
                                        their p_i probability
    wait for a data
   for i=0 to n
      if the data passed through slice i then
         X ← 1
      else
         X ← 0
      end if
      Generate R a x̃_i-Bernoulli random variable
      x̃_i ← x̃_i + 1/NbrLoop (X − R)
      Increment NbrLoop by one.
   end for
end repeat
```

Randomized Energy Balance Algorithms in Sensor Networks, Figure 2
Pseudo-code for estimation of the x_i value by the sink

Key Results

In [1,2] recursive equations similar to (2) were suggested and solved in closed form under adequate hypotheses. The need for a priori knowledge of the probability of occurrence of the events, the λ_i parameters, was considered in [7], in which these parameters were estimated by the sink on the basis of the observations of the various paths the data follow. The algorithm suggested is based on recursive estimation, is computationally not expensive and converges with rate $\mathcal{O}(1/\sqrt{n})$. One might argue that the rate of convergence is slow; however, it is numerically observed that relatively quickly compared with the convergence time, the algorithm finds an estimation close enough to the final value. The estimation algorithm run by the sink (which has no energy constraints) is given in Fig. 2.

Results taken from [1,2,7] all assume the existence of an energy-balance solution. However, particular distributions of the events might prevent the existence of such a solution and the relevant question is no longer the computation of an energy-balance algorithm. For instance, assuming that $\lambda_N = 0$, sensors in slice N have no way of balancing energy. In [9] the problem was reformulated as finding the probability distribution $\{p_i\}_{i=1,\ldots,N}$ which leads to the maximal functional lifetime of the networks. It was proved that if an energy-balance strategy exists, then it maximizes the lifetime of the network establishing formally the intuitive reasoning which was the motivation

to consider energy-balance strategies. A centralized algorithm was presented to compute the optimal parameters. Moreover, it was observed numerically that the interslice energy consumption is prone to be uneven and a spreading technique was suggested and numerically validated as being efficient to overcome this limitation of the probabilistic algorithm.

The communication graph considered is a restrictive subset of the complete communication graph and it is legitimate to wonder whether one can improve the situation by extending it. For instance, by allowing data to be sent two hops or more away. In [3,6] it was proved that the topology in which sensors communicate only to neighbor slices and the sink is the one which maximizes the flow of data in the network. Moreover, the communication graph in which sensors send data only to their neighbors and the sink leads to a completely distributed algorithm balancing energy [6]. Indeed, as a sensor sends data to a neighbor slice, the neighbor must in turn send the data and can attach information concerning its own energy level. This information might be captured by the initial sensor since it belongs to the communication range of its neighbor (this does not hold any longer if multiple hops are allowed). Hence, a distributed strategy consists in sending data to a particular neighbor only if its energy level consumption is lower, otherwise the data are sent directly to the sink.

Applications

Among the several constraints sensor networks designers have to face, energy management is central since sensors are usually battery powered, making the lifetime of the networks highly sensitive to the energy management. Besides the traditional strategy consisting in minimizing the energy consumption at sensor nodes, energy-balance schemes aim at balancing the energy consumption among sensors. The intuitive function of such schemes is to avoid energy depletion holes appearing as some sensors that run out of their available energy resources and are no longer able to participate in the global function of the networks. For instance, routing might be no longer possible if a small number of sensors run out of energy, leading to a disconnected network. This was pointed out in [5] as well as the need to develop application-specific protocols. Energy balancing is suggested as a solution in order to make the global functional lifetime of the network longer. The earliest development of dedicated protocols ensuring energy balance can be found in [4,10,11].

A key application is to maximize the lifetime of the network while gathering data to a sink. Besides increasing the lifetime of the networks, other criteria have to be taken into account. Indeed, the distributed algorithm might be as simple as possible owing to limited computational resources, might avoid collisions or limit the total number of transmissions, and might ensure a large enough flow of data from the sensors toward the sink. Actually, maximizing the flow of data is equivalent to maximizing the lifetime of sensor networks if some particular realizable conditions are fulfilled. Besides the simplicity of the distributed algorithm, the network deployment and the self-realization of the network structure might be possible in realistic conditions.

Cross References

- ► Obstacle Avoidance Algorithms in Wireless Sensor Networks
- ► Probabilistic Data Forwarding in Wireless Sensor Networks

Recommended Reading

1. Efthymiou, C., Nikoletseas, S., Rolim, J.: Energy Balanced Data Propagation in Wireless Sensor Networks. 4th International Workshop on Algorithms for Wireless, Mobile, Ad-Hoc and Sensor Networks (WMAN '04) IPDPS 2004, Wirel. Netw. J. (WINET) **12**(6), 691–707 (2006)
2. Efthymiou, C., Nikoletseas, S., Rolim, J.: Energy Balanced Data Propagation in Wireless Sensor Networks. In: Wireless Networks (WINET) Journal, Special Issue on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks. Springer (2006)
3. Giridhar, A., Kumar, P.R.: Maximizing the Functional Lifetime of Sensor Networks. In: Proceedings of The Fourth International Conference on Information Processing in Sensor Networks, IPSN '05, UCLA, Los Angeles, April 25–27 2005
4. Guo, W., Liu, Z., Wu, G.: An Energy-Balanced Transmission Scheme for Sensor Networks. In: 1st ACM International Conference on Embedded Networked Sensor Systems (ACM SenSys 2003), Poster Session, Los Angeles, CA, November 2003
5. Heinzelman, W., Chandrakasan, A., Balakrishnan, H.: Energy-efficient communication protocol for wireless microsensor networks. In: Proceedings of the 33rd IEEE Hawaii International Conference on System Sciences (HICSS 2000). 2000
6. Jarry, A., Leone, P., Powell, O., Rolim, J.: An Optimal Data Propagation Algorithm for Maximizing the Lifespan of Sensor Networks. In: Second International Conference, DCOSS 2006, San Francisco, CA, USA, June 2006. Lecture Notes in Computer Science, vol. 4026, pp. 405–421. Springer, Berlin (2006)
7. Leone, P., Nikoletseas, S., Rolim, J.: An Adaptive Blind Algorithm for Energy Balanced Data Propagation in Wireless Sensor Networks. In: First International Conference on Distributed Computing in Sensor Systems (DCOSS), Marina del Rey, CA, USA, June/July 2005. Lecture Notes in Computer Science, vol. 3560, pp. 35–48. Springer, Berlin (2005)
8. Olariu, S., Stojmenovic, I.: Design guidelines for maximizing lifetime and avoiding energy holes in sensor networks with uniform distribution and uniform reporting. In: IEEE INFOCOM, Barcelona, Spain, April 24–25 2006
9. Powell, O., Leone, P., Rolim, J.: Energy Optimal Data Propagation in Sensor Networks. J. Prarallel Distrib. Comput. **67**(3), 302–317 (2007) http://arxiv.org/abs/cs/0508052

10. Singh, M., Prasanna, V.: Energy-Optimal and Energy-Balanced Sorting in a Single-Hop Wireless Sensor Network. In: Proc. First IEEE International Conference on Pervasive Computing and Communications (PerCom '03), pp. 302–317, Fort Worth, 23–26 March 2003
11. Yu, Y., Prasanna, V.K.: Energy-Balanced Task Allocation for Collaborative Processing in Networked Embedded System. In: Proceedings of the 2003 Conference on Language, Compilers, and Tools for Embedded Systems (LCTES'03), pp. 265–274, San Diego, 11–13 June 2003

Randomized Gossiping in Radio Networks

2001; Chrobak, Gąsieniec, Rytter

Leszek Gąsieniec
Department of Computer Science,
University of Liverpool, Liverpool, UK

Keywords and Synonyms

Wireless networks; Broadcast; Gossip; Total exchange of information; All-to-all communication

Problem Definition

The two classical problems of disseminating information in computer networks are *broadcasting* and *gossiping*. In broadcasting, the goal is to distribute a message from a distinguished *source* node to all other nodes in the networks. In gossiping, each node v in the network initially contains a message m_v, and the task is to distribute each message m_v to all nodes in the network.

The radio network abstraction captures the features of distributed communication networks with multi-access channels, with minimal assumptions on the channel model and processors' knowledge. Directed edges model uni-directional links, including situations in which one of two adjacent transmitters is more powerful than the other. In particular, there is no feedback mechanism (see, for example, [6]). In some applications, collisions may be difficult to distinguish from the noise that is normally present in the channel, justifying the need for protocols that do not depend on the reliability of the collision detection mechanism (see [3,4]). Some network configurations are subject to frequent changes. In other networks, a network topology could be unstable or dynamic; for example, when mobile users are present. In such situations, algorithms that do not assume any specific topology are more desirable.

More formally a radio network is a directed graph $G = (V, E)$, where by $|V| = n$ we denote the number of nodes in this graph. Individual nodes in V are denoted by letters $u, v, \ldots$. If there is an edge from u to v, i. e., $(u, v) \in E$, then we say that v is an *out-neighbor* of u and u is an *in-neighbor* of v. Messages are denoted by letter m, possibly with indices. In particular, the message originating from node v is denoted by m_v. The whole set of initial messages is $M = \{m_v : v \in V\}$. During the computation, each node v holds a set of messages M_v that have been received by v so far. Initially, each node v does not possess any information apart from $M_v = \{m_v\}$. Without loss of generality, whenever a node is in the transmitting mode, one can assume that it transmits the whole content of M_v.

The time is divided into discrete time steps. All nodes start simultaneously, have access to a common clock, and work synchronously. A gossiping algorithm is a protocol that for each node u, given all past messages received by u, specifies, for each time step t, whether u will transmit a message at time t, and if so, it also specifies the message. A message M transmitted at time t from a node u is sent instantly to all its out-neighbors. An out-neighbor v of u receives M at time step t only if no collision occurred, that is, if the other in-neighbors of v do not transmit at time t at all. Further, collisions cannot be distinguished from background noise. If v does not receive any message at time t, it knows that either none of its in-neighbors transmitted at time t, or that at least two did, but it does not know which of these two events occurred. The *running time* of a gossiping algorithm is the smallest t such that for any network topology, and any assignment of identifiers to the nodes, all nodes receive messages originating in every other node no later than at step t.

Limited Broadcast$_v$(k) Given an integer k and a node v, the goal of *limited broadcasting* is to deliver the message m_v (originating in v) to at least k other nodes in the network.

Distributed Coupon Collection The set of network nodes V can be interpreted as a set of n bins and the set of messages M as a set of n coupons. Each coupon has at least k copies, each copy belonging to a different bin. M_v is the set of coupons in bin v. Consider the following process. At each step, one opens every bin at random, independently, with probability $1/n$. If no bin is opened, or if two or more bins are opened, a failure occurs and no coupons are collected. If exactly one bin, say v, is opened, all coupons from M_v are collected. The task is to establish how many steps are needed to collect (a copy of) each coupon.

Key Results

Theorem 1 ([1]) *There exists a deterministic $O(k \log^2 n)$-time algorithm for limited broadcasting from any node in radio networks with an arbitrary topology.*

Theorem 2 ([1]) *Let δ be a given constant, $0 < \delta < 1$, and $s = (4n/k)\ln(n/\delta)$. After s steps of the distributed coupon collection process, with probability at least $1 - \delta$, all coupons will be collected.*

Theorem 3 ([1]) *Let ϵ be a given constant, where $0 < \epsilon < 1$. There exists a randomized $O(n \log^3 n \log(n/\epsilon))$-time Monte Carlo-type algorithm that completes radio gossiping with probability at least $1 - \epsilon$.*

Theorem 4 ([1]) *There exists a randomized Las Vegas-type algorithm that completes radio gossiping with expected running time $O(n \log^4 n)$.*

Applications

Further work on efficient randomized radio gossiping include the $O(n \log^3 n)$-time algorithm by Liu and Prabhakaran, see [5], where the deterministic procedure for limited broadcasting is replaced by its $O(k \log n)$-time randomized counterpart. This bound was later reduced to $O(n \log^2 n)$ by Czumaj and Rytter in [2], where a new randomized limited broadcasting procedure with an expected running time $O(k)$ is proposed.

Open Problems

The exact complexity of randomized radio gossiping remains an open problem. All three gossiping algorithms [1,2,5] are based on the concepts of limited broadcast and distributed coupon collection. The two improvements [2,5] refer solely to limited broadcasting. Thus, very likely further reduction of the time complexity must coincide with more accurate analysis of the distributed coupon collection process or with development of a new gossiping procedure.

Recommended Reading

1. Chrobak, M., Gąsieniec, L., Rytter, W.: A Randomized Algorithm for Gossiping in Radio Networks. In: Proc. 8th Annual International Computing Combinatorics Conference. Guilin, China, pp. 483–492 (2001) Full version in Networks **43**(2), 119–124 (2004)
2. Czumaj, A., Rytter, W.: Broadcasting algorithms in radio networks with unknown topology. J. Algorithms **60**(2), 115–143 (2006)
3. Ephremides, A., Hajek, B.: Information theory and communication networks: an unconsummated union. IEEE Trans. Inf. Theor. **44**, 2416–2434 (1998)
4. Gallager, R.: A perspective on multiaccess communications. IEEE Trans. Inf. Theor. **31**, 124–142 (1985)
5. Liu, D., Prabhakaran, M.: On randomized broadcasting and gossiping in radio networks. In: Proc. 8th Annual International Computing Combinatorics Conference, pp. 340-349, Singapore (2002)
6. Massey, J.L., Mathys, P.: The collision channel without feedback. IEEE Trans. Inf. Theor. **31**, 192–204 (1985)

Randomized Minimum Spanning Tree

1995; Karger, Klein, Tarjan

Vijaya Ramachandran
Department of Computer Science,
University of Texas at Austin, Austin, TX, USA

Problem Definition

The input to the problem is a connected undirected graph $G = (V, E)$ with a weight $w(e)$ on each edge $e \in E$. The goal is to find a spanning tree of minimum weight, where for any subset of edges $E' \subseteq E$, the *weight of* E' is defined to be $w(E') = \sum_{e \in E'} w(e)$.

If the graph G is not connected, the goal of the problem is to find a *minimum spanning forest*, which is defined to be a minimum spanning tree in each connected component of G. Both problems will be referred to as the *MST* problem.

The randomized MST algorithm by Karger, Klein and Tarjan [9] which is considered here will be called the *KKT algorithm*. Also it will be assumed that the input graph $G = (V, E)$ has n vertices and m edges, and that the edge-weights are distinct.

The MST problem has been studied extensively prior to the KKT result, and several very efficient, deterministic algorithms are available from these studies. All of these are deterministic and are based on a method that greedily adds an edge to a forest that is a subgraph of the minimum spanning tree at all times. The early algorithms in this class are already efficient with a running time of $O(m \log n)$. These include the algorithms of Borůvka [1], Jarník [8] (later rediscovered by Dijkstra and Prim [5]) and Kruskal [5].

The fastest algorithm known for MST prior to the KKT algorithm runs in time $O(m \log \beta(m, n))$ [7], where $\beta(m, n) = \min\{i \mid \log^{(i)} n \leq m/n\}$ [7]; here $\log^{(i)} n$ is defined as $\log n$ if $i = 1$ and as $\log \log^{(i-1)} n$ if $i > 1$. Although this running time is close to linear, it is not linear-time if the graph is very sparse.

The problem of finding the minimum spanning tree efficiently is an important and fundamental problem in graph algorithms and combinatorial optimization.

Background

Some relevant background is summarized here.

- The basic step in Borůvka's algorithm [1] is the *Borůvka step*, which picks the minimum weight edge incident on each vertex, adds it to the minimum spanning tree, and then contracts these edges. This step runs in linear time, and also very efficiently in parallel. It is

the backbone of most efficient parallel algorithms for minimum spanning tree, and is also used in the KKT algorithm.

- A related and simpler problem is that of *minimum spanning tree verification*. Here, given a spanning tree T of the input edge-weighted graph, one needs to determine if T is its minimum spanning tree. An algorithm that solves this problem with a linear number of edge-weight comparisons was shown by Komlós [13], and later a deterministic linear-time algorithm was given in [6] (see also [12] for a simpler algorithm).

Key Results

The main result in [9] is a randomized algorithm for the minimum spanning tree problem that runs in expected linear time. The only operations performed on the edge-weights are pairwise comparisons. The algorithm does not assume any particular representation of the edge-weights (i. e., integer or real values), and only assumes that any comparison between a pair of edge-weights can be performed in unit time. The paper also shows that the algorithm runs in $O(m + n)$ time with the exponentially high probability $1 - exp(-\Omega(m))$, and that its worst-case running time is $O(n + m \log n)$.

The simple and elegant *MST sampling lemma* given in Lemma 1 below is the key tool used to derive and analyze the KKT algorithm. This lemma needs a couple of definitions and facts:

1. The well-known *cycle property* for minimum spanning tree states that the heaviest edge in any cycle in the input graph G *cannot* be in the minimum spanning tree.
2. Let F be a forest of G (i. e., an acyclic subgraph of G). An edge $e \in E$ is *F-light* if $F \cup \{e\}$ either continues to be a forest of G, or the heaviest edge in the cycle containing e is *not* e. An edge in G that is not F-light is *F-heavy*. Note that by the cycle property, an F-heavy edge cannot be in the minimum spanning tree of G, *no matter what forest F is used*. Given a forest F of G, the set of F-heavy edges can be determined in linear time by a simple modification to existing linear-time minimum spanning tree verification algorithms [6,12].

Lemma 1 (MST Sampling Lemma) *Let $H = (V, E_H)$ be formed from the input edge-weighted graph $G = (V, E)$ by including each edge with probability p independent of the other edges. Let F be the minimum spanning forest of H. Then, the expected number of F-light edges in G is $\leq n/p$.*

The KKT algorithm identifies edges in the minimum spanning tree of G only using Borůvka steps. However, after every two Borůvka steps, it removes F-heavy edges using the minimum spanning forest F of a subgraph obtained through sampling edges with probability $p = 1/2$. As mentioned earlier, these F-heavy edges can be identified in linear time. The minimum spanning forest of the sampled graph is computed recursively.

The correctness of the KKT algorithm is immediate since every F-heavy edge it removes cannot be in the MST of G since F is a forest of G, and every edge it adds to the minimum spanning tree is in the MST since it is added through a Borůvka step.

The expected running time analysis as well as the exponentially high probability bound for the running time are surprisingly simple to derive using the MST Sampling Lemma (Lemma 1).

In summary, the paper [9] proves the following results.

Theorem 2 *The KKT algorithm is a randomized algorithm that finds a minimum spanning tree of an edge-weighted undirected graph on n nodes and m edges in $O(n + m)$ time with probability at least $1 - \exp(-\Omega(m))$. The expected running time is $O(n + m)$ and the worst-case running time is $O(n + m \log n)$.*

The model of computation used in [9] is the unit-cost RAM model since the known MST verification algorithms were for this model, and not the more restrictive *pointer machine* model. More recently the MST verification result and hence the KKT algorithm have been shown to work on the pointer machine as well [2].

Lemma 1 is proved in [9] through a simulation of Kruskal's algorithm along with an analysis of the probability with which an F-light edge is not sampled. Another proof that uses a backward analysis is given in [3].

Further Comments

- Recently (and since the appearance of the KKT algorithm in 1995), two new deterministic algorithms for MST have appeared, due to Chazelle [4] and Pettie and Ramachandran [14]. The former [4] runs in $O(n + m\alpha(m, n))$ time, where α is an inverse of the Ackermann's function, whose growth rate is even smaller than the β function mentioned earlier for the best result that was known prior to the KKT algorithm [7]. The latter algorithm [14] provably runs in time that is within a constant factor of the decision-tree complexity of the MST problem, and hence is optimal; its time bound is $O(n + m\alpha(m, n))$ and $\Omega(n + m)$, and the exact bound remains to be determined.
- Although the KKT algorithm runs in expected linear time (and with exponentially high probability), it is not the last word on randomized MST algorithms. A randomized MST algorithm that runs in expected linear

time and uses only $O(\log^* n)$ random bits is given in [16,17]. In contrast, the KKT algorithm uses a linear number of random bits.

Applications

The minimum spanning tree problems has a large number of applications, which are discussed in Minimum spanning trees.

Open Problems

Some open problems that remain are the following:

1. Can randomness be removed in the KKT algorithm? A hybrid algorithm that uses the KKT algorithm within a modified version of the Pettie–Ramachandran algorithm [14] is given in [16,17] that achieves expected linear time while reducing the number of random bits used to only $O(\log^* n)$. Can this tiny amount of randomness be removed as well? If all randomness can be removed from the KKT algorithm, that will establish a linear time bound for the Pettie–Ramachandran algorithm [14] and also provide another optimal deterministic MST algorithm, this one based on the KKT approach.
2. Can randomness be removed from the work-optimal *parallel algorithms* [10] for MST? A linear-work, expected logarithmic-time parallel MST algorithm for the EREW PRAM is given in [15]. This parallel algorithm is both work- and time-optimal. However, it uses a linear number of random bits. Another work-optimal parallel algorithm is given in [16,17] that runs in expected polylog time using only polylog random bits. This leads to the following open questions regarding parallel algorithms for the MST problem:
   - To what extent can dependence on random bits be reduced (from the current linear bound) in a time- and work-optimal parallel algorithm for MST?
   - To what extent can the dependence on random bits be reduced (from the current polylog bound) in a work-optimal parallel algorithm with reasonable parallelism (say polylog parallel time)?

Experimental Results

Katriel, Sanders, and Träff [11] performed an experimental evaluation of the KKT algorithm and showed that it has good performance on moderately dense graphs.

Cross References

► Minimum Spanning Trees

Acknowledgments

This work was supported in part by NSF grant CFF-0514876.

Recommended Reading

1. Borůvka, O.: O jistém problému minimálním. Práce Moravské Přírodovědecké Společnosti **3**, 37–58 (1926) (In Czech)
2. Buchsbaum, A., Kaplan, H., Rogers, A., Westbrook, J.R.: Linear-time pointer-machine algorithms for least common ancestors, MST verification and dominators. In: Proc. ACM Symp. on Theory of Computing (STOC), 1998, pp. 279–288
3. Chan, T.M.: Backward analysis of the Karger–Klein–Tarjan algorithm for minimum spanning trees. Inf. Process. Lett. **67**, 303–304 (1998)
4. Chazelle, B.: A minimum spanning tree algorithm with inverse-Ackermann type complexity. J. ACM **47**(6), 1028–1047 (2000)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge (2001)
6. Dixon, B., Rauch, M., Tarjan, R.E.: Verification and sensitivity analysis of minimum spanning trees in linear time. SIAM J. Comput. **21**(6), 1184–1192 (1992)
7. Gabow, H.N., Galil, Z., Spencer, T.H., Tarjan, R.E.: Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Comb. **6**, 109–122 (1986)
8. Graham, R.L., Hell, P.: On the history of the minimum spanning tree problem. Ann. Hist. Comput. **7**(1), 43–57 (1985)
9. Karger, D.R., Klein, P.N., Tarjan, R.E.: A randomized linear-time algorithm for finding minimum spanning trees. J. ACM **42**(2), 321–329 (1995)
10. Karp, R.M., Ramachandran, V.: Parallel algorithms for shared-memory machines. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, pp. 869–941. Elsevier Science Publishers B.V., Amsterdam (1990)
11. Katriel, I., Sanders, P., Träff, J.L.: A practical minimum spanning tree algorithm using the cycle property. In: Proc. 11th Annual European Symposium on Algorithms. LNCS, vol. 2832, pp. 679–690. Springer, Berlin (2003)
12. King, V.: A simpler minimum spanning tree verification algorithm. Algorithmica **18**(2), 263–270 (1997)
13. Komlós, J.: Linear verification for spanning trees. Combinatorica **5**(1), 57–65 (1985)
14. Pettie, S., Ramachandran, V.: An optimal minimum spanning tree algorithm. J. ACM **49**(1), 16–34 (2002)
15. Pettie, S., Ramachandran, V.: A randomized time-work optimal parallel algorithm for finding a minimum spanning forest. SIAM J. Comput. **31**(6), 1879–1895 (2002)
16. Pettie, S., Ramachandran, V.: Minimizing randomness in minimum spanning tree, parallel connectivity, and set maxima algorithms. In: Proc. ACM-SIAM Symp. on Discrete Algorithms (SODA), 2002, pp. 713–722
17. Pettie, S., Ramachandran, V.: New randomized minimum spanning tree algorithms using exponentially fewer random bits. ACM Trans. Algorithms. **4**(1), article 5 (2008)

Randomized Parallel Approximations to Max Flow

1991; Serna, Spirakis

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

Keywords and Synonyms

Approximate maximum flow construction

Problem Definition

The work of Serna and Spirakis provides a parallel approximation schema for the Maximum Flow problem. An approximate algorithm provides a solution whose cost is within a factor of the optimal solution. The notation and definitions are the standard ones for networks and flows (see for example [2,7]).

A *network* $N = (G, s, t, c)$ is a structure consisting of a directed graph $G = (V, E)$, two distinguished vertices, $s, t \in V$ (called the *source* and the *sink*), and $c : E \to \mathbb{Z}^+$, an assignment of an integer capacity to each edge in E. A *flow function* f is an assignment of a non-negative number to each edge of G (called the flow into the edge) such that first at no edge does the flow exceed the capacity, and second for every vertex except s and t, the sum of the flows on its incoming edges equals the sum of the flows on its outgoing edges. The *total flow* of a given flow function f is defined as the net sum of flow into the sink t. The Maximum Flow problem can be stated as

Name Maximum Flow
Input A network $N = (G, s, t, c)$
Output Find a flow f for N for which the total flow is maximum.

Maximum Flows and Matchings The Maximum Flow problem is closely related to the Maximum Matching problem on bipartite graphs.

Given a graph $G = (V, E)$ and a set of edges $M \subseteq E$ is a *matching* if in the subgraph (V, M) all vertices have degree at most one. A *maximum matching* for G is a matching with a maximum number of edges. For a graph $G = (V, E)$ with weight $w(e)$, the *weight* of a matching M is the sum of the weights of the edges in M. The problem can be stated as follows:

Name Maximum Weight Matching
Input A graph $G = (V, E)$ and a weight $w(e)$ for each edge $e \in E$
Output Find a matching of G with the maximum possible weight.

There is a standard reduction from the Maximum Matching problem for bipartite graphs to the Maximum Flow problem ([7,8]). In the general weighted case one has just to look at each edge with capacity $c > 1$ as c edges joining the same points each with capacity one, and transform the multigraph obtained as shown before. Notice that to perform this transformation a c value is required which is polynomially bounded. The whole procedure was introduced by Karp, Upfal, and Wigderson [5] providing the following results

Theorem 1 *The Maximum Matching problem for bipartite graphs is NC equivalent to the Maximum Flow problem on networks with polynomial capacities. Therefore, the Maximum Flow with polynomial capacities problem belongs to the class RNC.*

Key Results

The first contribution is an extension of Theorem 1 to a generalization of the problem, namely the Maximum Flow on networks with polynomially bounded maximum flow. The proof is based on the construction (in NC) of a second network which has the same maximum flow but for which the maximum flow and the maximum capacity in the network are polynomially related.

Lemma 2 *Let $N = (G, s, t, c)$. Given any integer k, there is an NC algorithm that decides whether $f(N) \geq k$ or $f(N) < km$.*

Since Lemma 2 applies even to numbers that are exponential in size, they get

Lemma 3 *Let $N = (G, s, t, c)$ be a network, there is an NC algorithm that computes an integer value k such that $2^k \leq f(N) < m\, 2^{k+1}$.*

The following lemma establishes the NC-reduction from the Maximum Flow problem with polynomial maximum flow to the Maximum Flow problem with polynomial capacities.

Lemma 4 *Let $N = (G, s, t, c)$ be a network, there is an NC algorithm that constructs a second network $N_1 = (G, s, t, c_1)$ such that*

$$\log(\mathrm{Max}(N_1)) \leq \log(f(N_1)) + O(\log n)$$

and $f(N) = f(N_1)$.

Lemma 4 shows that the Maximum Flow problem restricted to networks with polynomially bounded maximum flow is NC-reducible to the Maximum Flow problem restricted to polynomially bounded capacities, the latter problem is a simplification of the former one, so the following results follow.

Theorem 5 *For each polynomial p, the problem of constructing a maximum flow in a network N such that $f(N) \leq p(n)$ is NC-equivalent to the problem of constructing a maximum matching in a bipartite graph, and thus it is in RNC.*

Recall that [5] gave us an $O(\log^2 n)$ randomized parallel time algorithm to compute a maximum matching. The combination of this with the reduction from the Maximum Flow problem to the Maximum Matching leads to the following result.

Theorem 6 *There is a randomized parallel algorithm to construct a maximum flow in a directed network, such that the number of processors is bounded by a polynomial in the number of vertices and the time used is* $O((\log n)^{\alpha} \log f(N))$ *for some constant* $\alpha > 0$.

The previous theorem is the first step towards finding an approximate maximum flow in a network N by an RNC algorithm. The algorithm, given N and an $\varepsilon > 0$, outputs a solution f' such that $f(N)/f' \leq 1 + 1/\varepsilon$. The algorithm uses a polynomial number of processors (independent of ε) and parallel time $O(\log^{\alpha} n(\log n + \log \varepsilon))$, where α is independent of ε. Thus, the algorithm is an RNC one as long as ε is at most polynomial in n. (Actually ε can be $O(n^{\log^{\beta} n})$ for some β.) Thus, being a Fully RNC approximation scheme (FRNCAS).

The second ingredient is a rough NC approximation to the Maximum Flow problem.

Lemma 7 *Let* $N = (G, s, t, c)$ *be a network. Let* $k \geq 1$ *be an integer, then there is an NC algorithm to construct a network* $M = (G, s, t, c_1)$ *such that* $k f(M) \leq f(N) \leq k f(M) + km$.

Putting all together and allowing randomization the algorithm can be sketched as follows:

FAST-FLOW($N = (G, s, t, c), \varepsilon$)

1. Compute k such that $2^k \leq F(N) \leq 2^{k+1} m$.
2. Construct a network N_1 such that

$$\log(\mathrm{Max}(N_1)) \leq \log(F(N_1)) + O(\log n) .$$

3. If $2^k \leq (1 + \varepsilon)m$ then $F(N) \leq (1 + \varepsilon)m^2$ so use the algorithm given in Theorem 6 to solve the Maximum Flow problem in N as a Maximum Matching and **return**
4. Let $\beta = \lfloor (2^k)/((1 + \varepsilon)m) \rfloor$. Construct N_2 from N_1 and β using the construction in Lemma 7.
5. Solve the Maximum Flow problem in N_2 as a Maximum Matching.
6. Output $F' = \beta F(M_2)$ and for all $e \in E$, $f'(e) = \beta f(e)$.

Theorem 8 *Let* $N = (G, s, t, c)$ *be a network. Then, algorithm FAST-FLOW is an RNC algorithm such that for all* $\varepsilon > 0$ *at most polynomial in the number of network vertices, the algorithm computes a legal flow of value* f' *such that*

$$\frac{f(N)}{f'} \leq 1 + \frac{1}{\varepsilon} .$$

Furthermore, the algorithm uses a polynomial number of processors and runs in expected parallel time $O(\log^{\alpha} n(\log n + \log \varepsilon))$, *for some constant* α, *independent of* ε.

Applications

The *rounding/scaling* technique is used in general to deal with problems that are hard due to the presence of large weights in the problem instance. The technique modifies the problem instance in order to produce a second instance that has no large weights, and thus can be solved efficiently. The way in which a new instance is obtained consists of computing first an estimate of the optimal value (when needed) in order to discard unnecessary high weights. Then the weights are modified, scaling them down by an appropriate factor that depends on the estimation and the allowed error. The rounding factor is determined in such a way that the so-obtained instance can be solved efficiently. Finally, a last step consisting of scaling up the value of the "easy" instance solution is performed in order to meet the corresponding accuracy requirements.

It is known that in the sequential case, the only way to construct FPTAS uses rounding/scaling and interval partition [6]. In general, both techniques can be paralyzed, although sometimes the details of the parallelization are non-trivial [1].

The Maximum Flow problem has a long history in Computer Science. Here are recorded some results about its parallel complexity. Goldschlager, Shaw, and Staples showed that the Maximum Flow problem is P-complete [3]. The P-completeness proof for Maximum Flow uses large capacities on the edges; in fact the values of some capacities are exponential in the number of network vertices. If the capacities are constrained to be no greater than some polynomial in the number of network vertices the problem is in ZNC. In the case of planar networks it is known that the Maximum Flow problem is in NC, even if arbitrary capacities are allowed [4].

Open Problems

The parallel complexity of the Maximum Weight Matching problem when the weight of the edges are given in binary is still an open problem. However, as mentioned earlier, there is a randomized NC algorithm to solve the problem in $O(\log^2 n)$ parallel steps, when the weights of the edges are given in unary. The scaling technique has been used to obtain fully randomized NC approximation schemes, for the Maximum Flow and Maximum Weight Matching problems (see [10]). The result appears to be the best possible in regard of full approximation, in the sense

that the existence of an FNCAS for any of the problems considered is equivalent to the existence of an NC algorithm for perfect matching which is also still an open problem.

Cross References

- ► Approximate Maximum Flow Construction
- ► Maximum Matching
- ► Paging

Recommended Reading

1. Díaz, J., Serna, M., Spirakis, P.G., Torán, J.: Paradigms for fast parallel approximation. In: Cambridge International Series on Parallel Computation, vol 8, Cambridge University Press, Cambridge (1997)
2. Even, S.: Graph Algorithms. Computer Science Press, Potomac (1979)
3. Goldschlager, L.M., Shaw, R.A., Staples, J.: The maximum flow problem is log-space complete for P. Theor. Comput. Sci. **21**, 105–111 (1982)
4. Johnson, D.B., Venkatesan, S.M.: Parallel algorithms for minimum cuts and maximum flows in planar networks. J. ACM **34**, 950–967 (1987)
5. Karp, R.M., Upfal, E., Wigderson, A.: Constructing a perfect matching is in Random NC. Combin. **6**, 35–48 (1986)
6. Korte, B., Schrader, R.: On the existence of fast approximation schemes. Nonlinear Program. **4**, 415–437 (1980)
7. Lawler, E.L.: Combinatorial Optimization: Networks and Matroids. Holt, Rinehart and Winston, New York (1976)
8. Papadimitriou, C.: Computational Complexity. Addison-Wesley, Reading (1994)
9. Peters, J.G., Rudolph, L.: Parallel aproximation schemes for subset sum and knapsack problems. Acta Inform. **24**, 417–432 (1987)
10. Spirakis, P.: PRAM models and fundamental parallel algorithm techniques: Part II. In: Gibbons, A., Spirakis, P. (eds.) Lectures on Parallel Computation, pp. 41–66. Cambrige University Press, New York (1993)

Randomized Rounding

1987; Raghavan, Thompson

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

Problem Definition

Randomized rounding is a technique for designing approximation algorithms for NP-hard optimization problems. Many combinatorial optimization problems can be represented as 0-1 integer linear programs; that is, integer linear programs in which variables take values in $\{0, 1\}$. While 0-1 integer linear programming is NP-hard, the rational relaxations (also referred to as fractional relaxations) of these linear programs are solvable in polynomial time [12,13]. Randomized rounding is a technique to construct a provably good solution to a 0-1 integer linear program from an optimum solution to its rational relaxation by means of a randomized algorithm.

Let Π be a 0-1 integer linear program with variables $x_i \in \{0, 1\}$, $1 \leq i \leq n$. Let Π_R be the rational relaxation of Π obtained by replacing the $x_i \in \{0, 1\}$ constraints by $x_i \in [0, 1]$, $1 \leq i \leq n$. The randomized rounding approach consists of two phases:

1. Solve Π_R using an efficient linear program solver. Let the variable x_i take on value $x_i^* \in [0, 1]$, $1 \leq i \leq n$.
2. Compute a solution to Π by setting the variables x_i randomly to one or zero according to the following rule:

$$\Pr[x_i = 1] = x_i^* .$$

For several fundamental combinatorial optimization problems, the randomized rounding technique yields simple randomized approximation algorithms that yield solutions provably close to optimal. Variants of the basic approach outlined above, in which the rounding of variable x_i in the second phase is done with a probability that is some appropriate function of x_i^*, have also been studied. The analyses of algorithms based on randomized rounding often rely on Chernoff–Hoeffding bounds from probability theory [5,11].

The work of Raghavan and Thompson [14] introduced the technique of randomized rounding for designing approximation algorithms for NP-hard optimization problems. The randomized rounding approach also implicitly proves the existence of a solution with certain desirable properties. In this sense, randomized rounding can be viewed as a variant of the probabilistic method, due to Erdös [1], which is widely used for various existence proofs in combinatorics.

Raghavan and Thompson illustrate the randomized rounding approach using three optimization problems: VLSI routing, multicommodity flow, and k-matching in hypergraphs.

Definition 1 In the **VLSI Routing** problem, we are given a two-dimensional rectilinear lattice L_n over n nodes and a collection of m *nets* $\{a_i : 1 \leq i \leq m\}$, where net a_i, is a set of nodes to be connected by means of a Steiner tree in L_n. For each net a_i, we are also given a set $\mathcal{A}_i$ of *allowed* trees that can be used for connecting the nodes in that set. A solution to the problem is a set $\mathcal{T}$ of trees $\{T_i \in \mathcal{A}_i : 1 \leq i \leq m\}$. The *width* of solution $\mathcal{T}$ is the maximum, over all edges e, of the number of trees in $\mathcal{T}$

that contain the edge. The goal of the VLSI routing problem is to determine a solution with minimum width.

Definition 2 In the **Multicommodity Flow Congestiom Minimization** problem (or simply, the Congestion Minimization problem), we are given a graph $G = (V, E)$, and a set of source-destination pairs $\{(s_i, t_i): 1 \leq i \leq k\}$. For each pair (s_i, t_i), we would like to route one unit of demand from s_i to t_i. A solution to the problem is a set $\mathcal{P} = \{P_i: 1 \leq i \leq k\}$ such that P_i is a path from s_i to t_i in G. We define the *congestion* of $\mathcal{P}$ to be the maximum, over all edges e, of the number of paths containing e. The goal of the undirected multicommodity flow problem is to determine a path set $\mathcal{P}$ with minimum congestion.

In their original work [14], Raghavan and Thompson studied the above problem for the case of undirected graphs and referred to it as the Undirected Multicommodity Flow problem. Here, we adopt the more commonly-used term of Congestion Minimization and consider both undirected and directed graphs since the results of [14] apply to both classes of graphs. Researchers have studied a number of variants of the multicommodity flow problem, which differ in various aspects of the problem such as the nature of demands (e. g., uniform vs. non-uniform), the objective function (e. g., the total flow vs. the maximum fraction of each demand), and edge capacities (e. g., uniform vs. non-uniform).

Definition 3 In the **Hypergraph Simple *k*-Matching** problem, we are given a hypergraph H over an n-element vertex set V. A k-matching of H is a set M of edges such that each vertex in V belongs to at most k of the edges in M. A k-matching M is simple if no edge in H occurs more than once in M. The goal of the problem is to determine a maximum-size simple k-matching of a given hypergraph H.

Key Results

Raghavan and Thompson present approximation algorithms for the above three problems using randomized rounding. In each case, the algorithm is easy to present: write a 0-1 integer linear program for the problem, solve the rational relaxation of this program, and then apply randomized rounding. They establish bounds on the quality of the solutions (i. e., the approximation ratios of the algorithm) using Chernoff–Hoeffding bounds on the tail of the sums of bounded and independent random variables [5,11].

The VLSI Routing problem can be easily expressed as a 0-1 integer linear program, say Π_1. Let W^* denote the width of the optimum solution to the rational relaxation of Π_1.

Theorem 1 *For any ε such that $0 < \varepsilon < 1$, the width of the solution produced by randomized rounding does not exceed*

$$W^* + \left[3W^* \ln \frac{2n(n-1)}{\varepsilon}\right]^{1/2}$$

with probability at least $1 - \varepsilon$, provided $W^* \geq 3\ln(2n(n-1)/\varepsilon)$.

Since W^* is a lower bound on the width of an optimum solution to Π_1, it follows that the randomized rounding algorithm has an approximation ratio of $1 + o(1)$ with high probability as long as W^* is sufficiently large.

The Congestion Minimization problem can be easily expressed as a 0-1 integer linear program, say Π_2. Let C^* denote the congestion of the optimum solution to the linear relaxation of Π_2. This optimum solution yields a set of flows, one for each commodity i. The flow for commodity i can be decomposed into a set Γ_i of at most $|E|$ paths from s_i to t_i. The randomized rounding algorithm selects, for each commodity i, one path P_i at random from Γ_i according to the flow values determined by the flow decomposition.

Theorem 2 *For any ε such that $0 < \varepsilon < 1$, the capacity of the solution produced by randomized rounding does not exceed*

$$C^* + \left[3C^* \ln \frac{|E|}{\varepsilon}\right]^{1/2}$$

with probability at least $1 - \varepsilon$, provided $C^* \geq 2\ln|E|$.

Since C^* is a lower bound on the width of an optimum solution to Π_1, it follows that the randomized rounding algorithm achieves a constant approximation ratio with probability $1 - 1/n$ when C^* is $\Omega(\log n)$.

For both the VLSI Routing and the Congestion Minimization problems, slightly worse approximation ratios are achieved if the lower bound condition on W^* and C^*, respectively, is removed. In particular, the approximation ratio achieved is $O(\log n/\log\log n)$ with probability at least $1 - n^{-c}$ for a constant $c > 0$ whose value depends on the constant hidden in the big-Oh notation.

The hypergraph k-matching problem is different than the above two problems in that it is a packing problem with a maximization objective while the latter are covering problems with a minimization objective. Raghavan and Thompson show that randomization rounding, in conjunction with a scaling technique, yields good approximation algorithms for the hypergraph k-matching problem.

They first express the matching problem as a 0-1 integer linear program, solve its rational relaxation Π_3, and then round the optimum rational solution by using appropriately scaled values of the variables as probabilities. Let S^* denote the value of the optimum solution to Π_3.

Theorem 3 *Let δ_1 and δ_2 be positive constants such that $\delta_2 > n \cdot e^{-k/6}$ and $\delta_1 + \delta_2 < 1$. Let $\alpha = 3\ln(n/\delta_2)/k$ and*

$$S' = S^* \left(1 - \frac{(\alpha^2 + 4\alpha)^{1/2} - \alpha}{2}\right).$$

Then, there exists a simple k-matching for the given hypergraph with size at least

$$S' - \left(2S' \ln \frac{1}{\delta_1}\right)^{1/2}.$$

Note that the above result is stated as an existence result. It can be modified to yield a randomized algorithm that achieves essentially the same bound with probability $1 - \varepsilon$ for a given failure probability ε.

Applications

Randomized rounding has found applications for a wide range of combinatorial optimization problems. Following the work of Raghavan and Thompson [14], Goemans and Williamson showed that randomized rounding yields an $e/(e-1)$-approximation algorithm for MAXSAT, the problem of finding an assignment that satisfies the maximum number of clauses of a given Boolean formula [7]. For the set cover problem, randomized rounding yields an algorithm with an asymptotically optimal approximation ratio of $O(\log n)$, where n is the number of elements in the given set cover instance [10]. Srinivasan has developed more sophisticated randomized rounding approaches for set cover and more general covering and packing problems [15]. Randomized rounding also yields good approximation algorithms for several flow and cut problems, including variants of undirected multicommodity flow [9] and the multiway cut problem [4].

While randomized rounding provides a unifying approach to obtain approximation algorithms for hard optimization problems, better approximation algorithms have been designed for specific problems. In some cases, randomized rounding has been combined with other algorithms to yield better approximation ratios than previously known. For instance, Goemans and Williamson showed that the better of two solutions, one obtained by randomized rounding and the other obtained by an earlier algorithm due to Johnson, yields a 4/3 approximation for MAXSAT [7].

The work of Raghavan and Thompson applied randomized rounding to a solution obtained for the relaxation of a 0-1 integer program for a given problem. In recent years, more sophisticated approximation algorithms have been obtained by applying randomized rounding to semidefinite program relaxations of the given problem. Examples include the 0.87856-approximation algorithm for MAXCUT due to Goemans and Williamson [8] and an $O(\sqrt{\log n})$-approximation algorithm for the sparsest cut problem, due to Arora, Rao, and Vazirani [3].

An excellent reference for the above and other applications of randomized rounding in approximation algorithms is the text by Vazirani [16].

Open Problems

While randomized rounding has yielded improved approximation algorithms for a number of NP-hard optimization problems, the best approximation achievable by a polynomial-time algorithm is still open for most of the problems discussed in this article, including MAXSAT, MAXCUT, the sparsest cut, the multiway cut, and several variants of the congestion minimization problem. For directed graphs, it has been shown that best approximation ratio achievable for congestion minimization in polynomial time is $\Omega(\log n/\log\log n)$, unless NP $\subset$ ZPTIME($n^{O(\log\log n)}$), matching the upper bound mentioned in Sect. "Key Results" up to constant factors [6]. For undirected graphs, the best known inapproximability lower bound is $\Omega(\log\log n/\log\log\log n)$ [2].

Cross References

► Oblivious Routing

Recommended Reading

1. Alon, N., Spencer, J.H.: The Probabilistic Method. Wiley, New York (1991)
2. Andrews, M., Zhang, L.: Hardness of the undirected congestion minimization problem. In: STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, pp. 284–293. ACM Press, New York (2005)
3. Arora, S., Rao, S., Vazirani, U.V.: Expander flows, geometric embeddings and graph partitioning. In: STOC, pp. 222–231. (2004)
4. Calinescu, G., Karloff, H.J., Rabani, Y.: An improved approximation algorithm for multiway cut. J. Comput. Syst. Sci. **60**(3), 564–574 (2000)
5. Chernoff, H.: A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations. Ann. Math. Stat. **23**, 493–509 (1952)
6. Chuzhoy, J., Guruswami, V., Khanna, S., Talwar, K.: Hardness of routing with congestion in directed graphs. In: STOC '07: Pro-

ceedings of the thirty-ninth annual ACM symposium on Theory of computing, pp. 165–178. ACM Press, New York (2007)
7. Goemans, M.X., Williamson, D.P.: New 3/4-approximation algorithms for the maximum satisfiability problem. SIAM J. Discret. Math. **7**, 656–666 (1994)
8. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM **42**(6), 1115–1145 (1995)
9. Guruswami, V., Khanna, S., Rajaraman, R., Shepherd, B., Yannakakis, M.: Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. J. Comput. Syst. Sci. **67**, 473–496 (2003)
10. Hochbaum, D.S.: Approximation algorithms for the set covering and vertex cover problems. SIAM J. Comput. **11**(3), 555–556 (1982)
11. Hoeffding, W.: On the distribution of the number of successes in independent trials. Ann. Math. Stat. **27**, 713–721 (1956)
12. Karmarkar, N.: A new polynomial-time algorithm for linear programming. Combinatorica **4**, 373–395 (1984)
13. Khachiyan, L.G.: A polynomial algorithm for linear programming. Soviet Math. Doklady **20**, 191–194 (1979)
14. Raghavan, P., Thompson, C.: Randomized rounding: A technique for provably good algorithms and algorithmic proofs. Combinatorica **7** (1987)
15. Srinivasan, A.: Improved approximations of packing and covering problems. In: Proceedings of the 27th Annual ACM Symposium on Theory of Computing, pp. 268–276 (1995)
16. Vazirani, V.: Approximation Algorithms. Springer (2003)

Randomized Searching on Rays or the Line

1993; Kao, Reif, Tate

Stephen R. Tate
University of North Carolina at Greensboro, Greensboro, NC, USA

Keywords and Synonyms

Cow-path problem; On-line navigation

Problem Definition

This problem deals with finding a point at an unknown position on one of a set of w rays which extend from a common point (the origin). In this problem there is a *searcher*, who starts at the origin, and follows a sequence of commands such as "explore to distance d on ray i." The searcher detects immdiately when the target point is crossed, but there is no other information provided from the search environment. The goal of the searcher is to minimize the distance traveled.

There are several different ways this problem has been formulated in the literature, including one called the "cow-path problem" that involves a cow searching for a pasture down a set of paths. When $w = 2$, this problem is to search for a point on the line, which has also been described as a robot searching for a door in an infinite wall or a shipwreck survivor searching for a stream after washing ashore on a beach.

Notation

The problem is as described above, with w rays. The position of the target point (or goal) is denoted (g, i) if it is at distance g on ray $i \in \{0, 1, \cdots, w-1\}$. The standard notion of *competitive ratio* is used when analyzing algorithms for this problem: An algorithm that knows which ray the goal is on will simply travel distance g down that ray before stopping, so search algorithms are compared to this optimal, omniscient strategy.

In particular, if $\mathcal{R}$ is a randomized algorithm, then the distance traveled to find a particular goal position is a random variable denoted $\mathsf{distance}(\mathcal{R}, (g, i))$, with expected value $E[\mathsf{distance}(\mathcal{R}, (g, i))]$. Algorithm $\mathcal{R}$ has competitive ratio c if there is a constant a such that, for all goal positions (g, i),

$$E[\mathsf{distance}(\mathcal{R}, (g, i))] \le c \cdot g + a\,. \tag{1}$$

Key Results

This problem is solved optimally using a randomized geometric sweep strategy: Search through the rays in a random (but fixed) order, with each search distance a constant factor longer than the preceding one. The initial search distance is picked from a carefully selected probability distribution, giving the following algorithm:

RaySearch$_{r,w}$
  $\sigma \leftarrow$ A random permutation of $\{0, 1, 2, \cdots, w-1\}$;
  $\epsilon \leftarrow$ A random real uniformly chosen from $[0, 1)$;
  $d \leftarrow r^{\epsilon}$;
  $p \leftarrow 0$;
  repeat
    Explore path $\sigma(p)$ up to distance d;
    if goal not found then return to origin;
    $d \leftarrow d \cdot r$;
    $p \leftarrow (p+1) \bmod w$;
  until goal found;

The theorems below give the competitive ratio of this algorithm, show how to pick the best r, and establish the optimality of the algorithm.

Theorem 1 ([9]) *For any fixed $r > 1$, Algorithm* RaySearch$_{r,w}$ *has competitive ratio*

$$R(r, w) = 1 + \frac{2}{w} \cdot \frac{1 + r + r^2 + \cdots + r^{w-1}}{\ln r}\,,$$

Randomized Searching on Rays or the Line, Table 1
The asymptotic growth of the competitive ratio with w is established in the following theorem

w	r_w^*	Optimal randomized ratio	Optimal deterministic ratio
2	3.59112	4.59112	9
3	2.01092	7.73232	14.5
4	1.62193	10.84181	19.96296
5	1.44827	13.94159	25.41406
6	1.35020	17.03709	30.85984
7	1.28726	20.13033	36.30277

Theorem 2 ([9]) *The unique solution of the equation*

$$\ln r = \frac{1 + r + r^2 + \cdots + r^{w-1}}{r + 2r^2 + 3r^3 + \cdots + (w-1)r^{w-1}} \tag{2}$$

for $r > 1$, denoted by r_w^*, gives the minimum value for $R(r, w)$.

Theorem 3 ([7,9,12]) *The optimal competitive ratio for any randomized algorithm for searching on w rays is*

$$\min_{r>1} \left\{ 1 + \frac{2}{w} \cdot \frac{1 + r + r^2 + \cdots + r^{w-1}}{\ln r} \right\} .$$

Corollary 1 *Algorithm* RaySearch$_{r,w}$ *is optimally competitive.*

Using Theorem 2 and standard numerical techniques, r_w^* can be computed to any required degree of precision. The following table shows, for small values of w, approximate values for r_w^* and the corresponding optimal competitive ratio (achieved by RaySearch$_{r,w}$)—the optimal deterministic competitive ratio (see [1]) is also shown for comparison:

Theorem 4 ([9]) *The competitive ratio for algorithm* RaySearch$_{r,w}$ *(with $r = r_w^*$) is $\kappa w + o(w)$, where*

$$\kappa = \min_{s>0} \left[2\frac{e^s - 1}{s^2} \right] \approx 3.088 .$$

Applications

The most direct applications of this problem are in geometric searching, such as robot navigation problems. For example, when a robot is traveling in an unknown area and encounters an obstacle, a typical first step is to find the nearest corner to go around [2,3], which is just an instance of the ray searching problem (with $w = 2$).

In addition, any abstract search problem with a cost function that is linear in the distance to the goal reduces to ray searching. This includes applications in artificial intelligence that search for a goal in a largely unknown search space [11] and the construction of hybrid algorithms [7]. In hybrid algorithms, a set of algorithms $A_1, A_2, \cdots, A_w$ for solving a problem is considered—algorithm A_1 is run for a certain amount of time, and if the algorithm is not successful algorithm A_1 is stopped and algorithm A_2 is started, repeating through all algorithms as many times as is necessary to find a solution. This notion of hybrid algorithms has been used successfully for several problems (such as the first competitive algorithm for the online k-server problem [4]), and the ray search algorithm gives the optimal strategy for selecting the trial running times of each algorithm.

Open Problems

Several natural extensions of this problem have been studied in both deterministic and randomized settings, including ray-searching when an upper bound on the distance to the goal is known (i. e., the rays are not infinite, but are line segments) [10,5,12], or when a probability distribution of goal positions is known [8]. Other variations of this basic searching problem have been studied for deterministic algorithms only, such as when the searcher's control is imperfect (so distances can't be specified precisely) [6] and for more general search spaces like points in the plane [1]. A thorough study of these variants with randomized algorithms remains an open problem.

Cross References

- Alternative Performance Measures in Online Algorithms
- Deterministic Searching on the Line
- Robotics

Recommended Reading

1. Baeza-Yates, R.A., Culberson, J.C., Rawlins, G.J.E.: Searching in the plane. Inf. Comput. **16**, 234–252 (1993)
2. Berman, P., Blum, A., Fiat, A., Karloff, H., Rosén, A., Saks, M.: Randomized robot navigation algorithms. In: Proceedings, Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 75–84 (1996)
3. Blum, A., Raghavan, P., Schieber, B.: Navigating in unfamiliar geometric terrain. In: Proceedings 23rd ACM Symposium on Theory of Computing (STOC), pp. 494–504 (1991)
4. Fiat, A., Rabani, Y., Ravid, Y.: Competitive k-server algorithms. In: Proceedings 31st IEEE Symposium on Foundations of Computer Science (FOCS), pp. 454–463 (1990)
5. Hipke, C., Icking, C., Klein, R., Langetepe, E.: How to find a point on a line within a fixed distance. Discret. Appl. Math. **93**, 67–73 (1999)

6. Kamphans, T., Langetepe, E.: Optimal competitive online ray search with an error-prone robot. In: 4th International Workshop on Experimental and Efficient Algorithms, pp. 593–596 (2005)
7. Kao, M., Ma, Y., Sipser, M., Yin, Y.: Optimal constructions of hybrid algorithms. In: Proceedings 5th ACM-SIAM Symposium on Discrete Algorithms (SODA) pp. 372–381 (1994)
8. Kao, M.-Y., Littman, M.L.: Algorithms for informed cows. In: AAAI-97 Workshop on On-Line Search, pp. 55–61 (1997)
9. Kao, M.-Y., Reif, J.H., Tate, S.R.: Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. Inf. Comput. **133**, 63–80 (1996)
10. López-Ortiz, A., Schuierer, S.: The ultimate strategy to search on m rays? Theor. Comput. Sci. **261**, 267–295 (2001)
11. Pearl, J.: Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, Reading, MA (1984)
12. Schuierer, S.: A lower bound for randomized searching on *m* rays. In: Computer Science in Perspective, pp. 264–277 (2003)

Random Number Generation

► Weighted Random Sampling

Random Planted 3-SAT

2003; Flaxman

Abraham Flaxman
Theory Group, Microsoft Research, Redmond, WA, USA

Keywords and Synonyms

Constraint satisfaction

Problem Definition

This classic problem in complexity theory is concerned with efficiently finding a satisfying assignment to a propositional formula. The input is a formula with n Boolean variables which is expressed as an AND of ORs with 3 variables in each OR clause (a *3-CNF formula*). The goal is to (1) find an assignment of variables to TRUE and FALSE so that the formula has value TRUE, or (2) prove that no such assignment exists. Historically, recognizing satisfiable 3-CNF formulas was the first "natural" example of an NP-complete problem, and, because it is NP-complete, no polynomial-time algorithm can succeed on all 3-CNF formulas unless P = NP [4,10]. Because of the numerous practical applications of 3-SAT, and also due to its position as the canonical NP-complete problem, many heuristic algorithms have been developed for solving 3-SAT, and some of these algorithms have been analyzed rigorously on random instances.

Notation A 3-CNF formula over variables $x_1, x_2, \ldots, x_n$ is the conjunction of m clauses $C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each clause is the disjunction of 3 literals, $C_i = \ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3}$, and each literal ℓ_{i_j} is either a variable or the negation of a variable (the negation of the variable x is denoted by $\overline{x}$). A 3-CNF formula is *satisfiable* if and only if there is an assignment of variables to truth values so that every clause contains at least one true literal. Here, all asymptotic analysis is in terms of n, the number of variables in the 3-CNF formula, and a sequence of events $\{\mathcal{E}_n\}$ is said to hold *with high probability* (abbreviated **whp**) if $\lim_{n\to\infty} \Pr[\mathcal{E}_n] = 1$.

Distributions There are many distributions over 3-CNF formulas which are interesting to consider, and this chapter focuses on dense satisfiable instances. Dense satisfiable instances can be formed by conditioning on the event $\{I_{n,m}$ is satisfiable$\}$, but this conditional distribution is difficult to sample from and to analyze. This has led to research in "planted" random instances of 3-SAT, which are formed by first choosing a truth assignment φ uniformly at random, and then selecting each clause independently from the triples of literals where at least one literal is set to TRUE by the assignment φ. The clauses can be included with equal probabilities in analogy to the $\mathbb{I}_{n,p}$ or $\mathbb{I}_{n,m}$ distributions above [8,9], or different probabilities can be assigned to the clauses with one, two, or three literals set to TRUE by φ, in an effort to better hide the satisfying assignment [2,7].

Problem 1 (3-SAT)
Input: *3-CNF Boolean formula $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each clause C_i is of the form $C_i = \ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3}$, and each literal ℓ_{i_j} is either a variable or the negation of a variable.*
Output: *A truth assignment of variables to Boolean values which makes at least one literal in each clause TRUE, or a certificate that no such assignment exists.*

Key Results

A line of basic research dedicated to identifying hard search and decision problems, as well as the potential cryptographic applications of planted instances of 3-SAT, has motivated the development of algorithms for 3-SAT which are known to work on planted random instances.

Majority Vote Heuristic: If every clause consistent with the planted assignment is included with the same probability, then there is a bias towards including the literal satisfied by the planted assignment more frequently than its negation. This is the motivation behind the Majority Vote Heuristic, which assigns each variable to the truth value which will satisfy the majority of the clauses in which it appears. Despite its simplicity, this heuristic has been proven successful **whp** for sufficiently dense planted instances [8].

Theorem 1 *When c is a sufficiently large constant and* $I \sim \mathbb{I}^{\phi}_{n,cn\log n}$, ***whp*** *the majority vote heuristic finds the planted assignment* φ.

When the density of the planted random instance is lower than $c \log n$, then the majority vote heuristic will fail, and if the relative probability of the clauses satisfied by one, two, and three literals are adjusted appropriately then it will fail miserably. But there are alternative approaches.

For planted instances where the density is a sufficiently large constant, the majority vote heuristic provides a good starting assignment, and then the *k-OPT heuristic* can finish the job. The *k*-OPT heuristic of [6] is defined as follows: Initialize the assignment by majority vote. Initialize k to 1. While there exists a set of k variables for which flipping the values of the assignment will (1) make false clauses true and (2) will not make true clauses false, flip the values of the assignment on these variables. If this reaches a local optimum that is not a satisfying assignment, increase k and continue.

Theorem 2 *When c is a sufficiently large constant and* $I \sim I^{\phi}_{n,cn}$ *the k-OPT heuristic finds a satisfying assignment in polynomial time* ***whp****. The same is true even in the semi-random case, where an adversary is allowed to add clauses to I that have all three literals set to TRUE by* φ *before giving the instance to the k-OPT heuristic.*

A related algorithm has been shown to run in expected polynomial time in [9], and a rigorous analysis of *Warning Propagation (WP)*, a message passing algorithm related to Survey Propagation, has shown that WP is successful **whp** on planted satisfying assignments, provided that the clause density exceeds a sufficiently large constant [5].

When the relative probabilities of clauses containing one, two, and three literals are adjusted carefully, it is possible to make the majority vote assignment very different from the planted assignment. A way of setting these relative probabilities that is predicted to be difficult is discussed in [2]. If the density of these instances is high enough (and the relative probabilities are anything besides the case of "Gaussian elimination with noise"), then a spectral heuristic provides a starting assignment close to the planted assignment and local reassignment operations are sufficient to recover a satisfying assignment [7].

More formally, consider instance $I = I_{n,p_1,p_2,p_3}$, formed by choosing a truth assignment φ on n variables uniformly at random and including in I each clause with exactly i literals satisfied by φ independently with probability p_i. By setting $p_1 = p_2 = p_3$ this reduces to the distribution mentioned above.

Setting $p_1 = p_2$ and $p_3 = 0$ yields a natural distribution on 3CNFs with a planted not-all-equal assignment, a situation where the greedy variable assignment rule generates a random assignment. Setting $p_2 = p_3 = 0$ gives 3CNFs with a planted exactly-one-true assignment (which succumb to the greedy algorithm followed by the non-spectral steps below). Also, correctly adjusting the ratios of p_1, p_2, and p_3 can obtain a variety of (slightly less natural) instance distributions which thwart the greedy algorithm. Carefully selected values of p_1, p_2, and p_3 are considered in [2], where it is conjectured that no algorithm running in polynomial time can solve I_{n,p_1,p_2,p_3} **whp** when $p_i = c_i\alpha/n^2$ and

$$0.077 < c_3 < 0.25 \qquad c_2 = (1-4c_3)/6$$
$$c_1 = (1+2c_3)/6 \qquad \alpha > \frac{4.25}{7}\,.$$

The *spectral heuristic* modeled after the coloring algorithms of [1,3] was developed for such planted distributions in [7]. This polynomial time algorithm which returns a satisfying assignment to I_{n,p_1,p_2,p_3} **whp** when $p_1 = d/n^2$, $p_2 = \eta_2 d/n^2$ and $p_3 = \eta_3 d/n^2$, for $0 \le \eta_2, \eta_3 \le 1$, and $d \ge d_{\min}$, where $d_{\min}$ is a function of η_2, η_3. The algorithm is structured as follows:

1. Construct a graph G from the 3CNF.
2. Find the most negative eigenvalue of a matrix related to the adjacency matrix of G.
3. Assign a value to each variable based on the signs of the eigenvector corresponding to the most negative eigenvalue.
4. Iteratively improve the assignment.
5. Perfect the assignment by exhaustive search over a small set containing all the incorrect variables.

A more elaborate description of each step is the following:

Step (1): Given 3CNF $I = I_{n,p_1,p_2,p_3}$, where $p_1 = d/n^2$, $p_2 = \eta_2 d/n^2$, and $p_3 = \eta_3 d/n^2$, the graph in step (1), $G = (V, E)$, has $2n$ vertices, corresponding to the literals in I, and labeled $\{x_1, \overline{x}_1, \ldots x_n, \overline{x}_n\}$. G has an edge between vertices ℓ_i and ℓ_j if I includes a clause with both ℓ_i and ℓ_j (and G does not have multiple edges).

Step (2): Consider $G' = (V, E')$, formed by deleting all the edges incident to vertices with degree greater than $180d$. Let A be the adjacency matrix of G'. Let λ be the most negative eigenvalue of A and $\mathbf{v}$ be the corresponding eigenvector.

Step (3): There are two assignments to consider, π_+, which is defined by

$$\pi_+(x_i) = \begin{cases} T\,, & \text{if} \quad \mathbf{v}_i \ge 0\,; \\ F\,, & \text{otherwise}\,; \end{cases}$$

and π_-, which is defined by

$$\pi_-(x) = \neg\pi_+(x) .$$

Let π_0 be the better of π_+ and π_- (that is, the assignment which satisfies more clauses). It can be shown that π_0 agrees with φ on at least $(1 - C/d)n$ variables for some absolute constant C.

Step (4): For $i = 1, \ldots, \log n$ do the following: for each variable x, if x appears in $5\varepsilon d$ clauses unsatisfied by π_{i-1}, then set $\pi_i(x) = \neg\pi_{i-1}(x)$, where ε is an appropriately chosen constant (taking $\varepsilon = 0.1$ works); otherwise set $\pi_i(x) = \pi_{i-1}(x)$.

Step (5): Let $\pi'_0 = \pi_{\log n}$ denote the final assignment generated in step (4). Let $\mathcal{A}_4^{\pi'_0}$ be the set of variables which do not appear in $(3 \pm 4\varepsilon)d$ clauses as the only true literal with respect to assignment π'_0, and let $\mathcal{B}$ be the set of variables which do not appear in $(\mu_D \pm \varepsilon)d$ clauses, where $\mu_D d = (3+6)d + (6+3)\eta_2 d + 3\eta_3 d + \mathcal{O}(1/n)$ is the expected number of clauses containing variable x. Form partial assignment $\pi_1{}'$ by unassigning all variables in $\mathcal{A}_4^{\pi'_0}$ and $\mathcal{B}$. Now, for $i \geq 1$, if there is a variable x_i which appears in less than $(\mu_D - 2\varepsilon)d$ clauses consisting of variables that are all assigned by π'_i, then let π'_{i+1} be the partial assignment formed by unassigning x_i in π'_i. Let π' be the partial assignment when this process terminates. Consider the graph Γ with a vertex for each variable that is unassigned in π' and an edge between two variables if they appear in a clause together. If any connected component in Γ is larger than $\log n$ then fail. Otherwise, find a satisfying assignment for I by performing an exhaustive search on the variables in each connected component of Γ.

Theorem 3 *For any constants $0 \leq \eta_2, \eta_3 \leq 1$, except $(\eta_2, \eta_3) = (0, 1)$, there exists a constant $d_{\min}$ such that for any $d \geq d_{min}$, if $p_1 = d/n^2$, $p_2 = \eta_2 d/n^2$, and $p_3 = \eta_3 d/n^2$ then this polynomial-time algorithm produces a satisfying assignment for random instances drawn from I_{n,p_1,p_2,p_3}* ***whp****.*

Applications

3-SAT is a universal problem, and due to its simplicity, it has potential applications in many areas, including proof theory and program checking, planning, cryptanalysis, machine learning, and modeling biological networks.

Open Problems

An important direction is to develop alternative models of random distributions which more accurately reflect the type of instances that occur in the real world.

Data Sets

Sample instances of satisfiability and 3-SAT are available on the web at http://www.satlib.org/.

URL to Code

Solvers and information on the annual satisfiability solving competition are available on the web at http://www.satlive.org/.

Recommended Reading

1. Alon, N., Kahale, N.: A spectral technique for coloring random 3-colorable graphs. SIAM J. Comput. **26**(6), 1733–1748 (1997)
2. Barthel, W., Hartmann, A.K., Leone, M., Ricci-Tersenghi, F., Weigt, M., Zecchina, R.: Hiding solutions in random satisfiability problems: A statistical mechanics approach. Phys. Rev. Lett. **88**, 188701 (2002)
3. Chen, H., Frieze, A.M.: Coloring bipartite hypergraphs. In: Cunningham, H.C., McCormick, S.T., Queyranne, M. (eds.) Integer Programming and Combinatorial Optimization, 5th International IPCO Conference, Vancouver, British Columbia, Canada, June 3–5 1996. Lecture Notes in Computer Science, vol. 1084, pp. 345–358. Springer
4. Cook, S.: The complexity of theorem-proving procedures. In: Proceedings of the 3rd Annual Symposium on Theory of Computing, pp. 151–158. Shaker Heights. May 3–5, 1971.
5. Feige, U., Mossel, E., Vilenchik, D.: Complete convergence of message passing algorithms for some satisfiability problems. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2006 and 10th International Workshop on Randomization and Computation, RANDOM 2006, Barcelona, Spain, August 28–30 2006. Lecture Notes in Computer Science, vol. 4110, pp. 339–350. Springer
6. Feige, U., Vilenchik, D.: A local search algorithm for 3-SAT, Tech. rep. The Weizmann Institute, Rehovat, Israel (2004)
7. Flaxman, A.D.: A spectral technique for random satisfiable 3CNF formulas. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (Baltimore, MD, 2003), pp. 357–363. ACM, New York (2003)
8. Koutsoupias, E., Papadimitriou, C.H.: On the greedy algorithm for satisfiability. Inform. Process. Lett. **43**(1), 53–55 (1992)
9. Krivelevich, M., Vilenchik, D.: Solving random satisfiable 3CNF formulas in expected polynomial time. In: SODA '06: Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorith. ACM, Miami, Florida (2006)
10. Levin, L.A.: Universal enumeration problems. Probl. Pereda. Inf. **9**(3), 115–116 (1973)

Ranked Matching

2005; Abraham, Irving, Kavitha, Mehlhorn

KAVITHA TELIKEPALLI
CSA Department, Indian Institute of Science, Bangalore, India

Keywords and Synonyms

Popular matching

Problem Definition

This problem is concerned with matching a set of *applicants* to a set of *posts*, where each applicant has a *preference list*, ranking a non-empty subset of posts in order of preference, possibly involving ties. Say that a matching M is *popular* if there is no matching M' such that the number of applicants preferring M' to M exceeds the number of applicants preferring M to M'. The ranked matching problem is to determine if the given instance admits a popular matching and if so, to compute one. There are many practical situations that give rise to such large-scale matching problems involving two sets of participants – for example, pupils and schools, doctors and hospitals – where participants of one set express preferences over the participants of the other set; an allocation determined by a popular matching can be regarded as an optimal allocation in these applications.

Notations and Definitions

An instance of the *ranked matching problem* is a bipartite graph $G = (\mathcal{A} \cup \mathcal{P}, E)$ and a partition $E = E_1 \dot\cup E_2 \ldots \dot\cup E_r$ of the edge set. Call the nodes in $\mathcal{A}$ *applicants*, the nodes in $\mathcal{P}$ *posts*, and the edges in E_i the edges of rank i. If $(a, p) \in E_i$ and $(a, p') \in E_j$ with $i < j$, say that a prefers p to p'. If $i = j$, say that a is indifferent between p and p'. An instance is *strict* if the degree of every applicant in every E_i is at most one.

A matching M is a set of edges, no two of which share an endpoint. In a matching M, a node $u \in \mathcal{A} \cup \mathcal{P}$ is either *unmatched*, or *matched* to some node, denoted by $M(u)$. Say that an applicant a *prefers* matching M' to M if (i) a is matched in M' and unmatched in M, or (ii) a is matched in both M' and M, and a prefers $M'(a)$ to $M(a)$.

Definition 1 M' is *more popular than* M, denoted by $M' \succ M$, if the number of applicants preferring M' to M exceeds the number of applicants preferring M to M'. A matching M is popular if and only if there is no matching M' that is more popular than M.

Figure 1 shows an instance with $A = \{a_1, a_2, a_3\}$, $P = \{p_1, p_2, p_3\}$, and each applicant prefers p_1 to p_2, and p_2 to p_3 (assume throughout that preferences are transitive). Consider the three symmetrical matchings $M_1 = \{(a_1, p_1), (a_2, p_2), (a_3, p_3)\}$, $M_2 = \{(a_1, p_3), (a_2, p_1), (a_3, p_2)\}$ and $M_3 = \{(a_1, p_2), (a_2, p_3), (a_3, p_1)\}$. It is easy to verify that none of these matchings is popular, since $M_1 \prec M_2$, $M_2 \prec M_3$, and $M_3 \prec M_1$. In fact, this instance admits no popular matching—the problem being, of course, that the *more popular than* relation is not acyclic, and so there need not be a maximal element.

a_1 :	p_1	p_2	p_3
a_2 :	p_1	p_2	p_3
a_3 :	p_1	p_2	p_3

Ranked Matching, Figure 1
An instance for which there is no popular matching

The *ranked matching problem* is to determine if a given instance admits a popular matching, and to find such a matching, if one exists. Popular matchings may have different sizes, and a largest such matching may be smaller than a maximum-cardinality matching. The *maximum-cardinality popular matching problem* then is to determine if a given instance admits a popular matching, and to find a *largest* such matching, if one exists.

Key Results

First consider *strict instances*, that is, instances $(\mathcal{A} \cup \mathcal{P}, E)$ where there are no ties in the preference lists of the applicants. Let n be the number of vertices and m be the number of edges in G.

Theorem 1 *For a strict instance $G = (\mathcal{A} \cup \mathcal{P}, E)$, it is possible to determine in $O(m + n)$ time if G admits a popular matching and compute one, if it exists.*

Theorem 2 *Find a maximum-cardinality popular matching of a strict instance $G = (\mathcal{A} \cup \mathcal{P}, E)$, or determine that no such matching exists, in $O(m + n)$ time.*

Next consider the general problem, where preference lists may have ties.

Theorem 3 *Find a popular matching of $G = (\mathcal{A} \cup \mathcal{P}, E)$, or determine that no such matching exists, in $O(\sqrt{n}m)$ time.*

Theorem 4 *Find a maximum-cardinality popular matching of $G = (\mathcal{A} \cup \mathcal{P}, E)$, or determine that no such matching exists, in $O(\sqrt{n}m)$ time.*

Techniques

Our results are based on a novel characterization of popular matchings. For exposition purposes, create a unique *last resort* post $l(a)$ for each applicant a and assign the edge $(a, l(a))$ a rank higher than any edge incident on a. In this

way, assume that every applicant is matched, since any unmatched applicant can be allocated to his/her last resort. From now on then, matchings are *applicant-complete*, and the size of a matching is just the number of applicants not matched to their last resort. Also assume that instances have no gaps, i. e., if an applicant has a rank i edge incident to it then it has edges of all smaller ranks incident to it. First outline the characterization in strict instances and then extend it to general instances.

Strict Instances For each applicant a, let $f(a)$ denote the most preferred post on a's preference list. That is, $(a, f(a)) \in E_1$. Call any such post p an *f-post*, and denote by $f(p)$ the set of applicants a for which $f(a) = p$.

For each applicant a, let $s(a)$ denote the most preferred non-f-post on a's preference list; note that $s(a)$ must exist, due to the introduction of $l(a)$. Call any such post p an *s-post*, and remark that f-posts are disjoint from s-posts.

Using the definitions of f-posts and s-posts, show three conditions that a popular matching must satisfy.

Lemma 5 *Let M be a popular matching.*

1. *For every f-post p, (i) p is matched in M, and (ii) $M(p) \in f(p)$.*
2. *For every applicant a, $M(a)$ can never be strictly between $f(a)$ and $s(a)$ on a's preference list.*
3. *For every applicant a, $M(a)$ is never worse than $s(a)$ on a's preference list.*

It is then shown that these three necessary conditions are also sufficient. This forms the basis of the following preliminary characterization of popular matchings.

Lemma 6 *A matching M is popular if and only if (i) every f-post is matched in M, and (ii) for each applicant a, $M(a) \in \{f(a), s(a)\}$.*

Given an instance graph $G = (\mathcal{A} \cup \mathcal{P}, E)$, define the *reduced graph* $G' = (\mathcal{A} \cup \mathcal{P}, E')$ as the subgraph of G containing two edges for each applicant a: one to $f(a)$, the other to $s(a)$. The authors remark that G' need not admit an applicant-complete matching, since $l(a)$ is now isolated whenever $s(a) \neq l(a)$. Lemma 6 shows that a matching is popular if and only if it belongs to the graph G' and it matches every f-post. Recall that all popular matchings are applicant-complete through the introduction of last resorts. Hence, the following characterization is immediate.

Theorem 7 *M is a popular matching of G if and only if (i) every f-post is matched in M, and (ii) M is an applicant-complete matching of the reduced graph G'.*

The characterization in Theorem 7 immediately suggests the following algorithm for solving the popular matching problem. Construct the reduced graph G'. If G' does not admit an applicant-complete matching, then G admits no popular matching. If G' admits an applicant-complete matching M, then modify M so that every f-post is matched. So for each f-post p that is unmatched in M, let a be any applicant in $f(p)$; remove the edge $(a, M(a))$ from M and instead match a to p. This algorithm can be implemented in $O(m + n)$ time. This shows Theorem 1.

Now, consider the maximum-cardinality popular matching problem. Let $\mathcal{A}_1$ be the set of all applicants a with $s(a) = l(a)$. Let $\mathcal{A}_1$ be the set of all applicants with $s(a) = l(a)$. Our target matching must satisfy conditions (i) and (ii) of Theorem 7, and among all such matchings, allocate the fewest $\mathcal{A}_1$-applicants to their last resort. This scheme can be implemented in $O(m + n)$ time. This proves Theorem 2.

General Instances For each applicant a, let $f(a)$ denote the *set* of first-ranked posts on a's preference list. Again, refer to all such posts p as *f-posts*, and denote by $f(p)$ the set of applicants a for which $p \in f(a)$. It may no longer be possible to match *every* f-post p with an applicant in $f(p)$ (as in Lemma 5), since, for example, there may now be more f-posts than applicants. Let M be a popular matching of some instance graph $G = (\mathcal{A} \cup \mathcal{P}, E)$. Define the *first-choice graph* of G as $G_1 = (\mathcal{A} \cup \mathcal{P}, E_1)$, where E_1 is the set of all rank one edges. Next the authors show the following lemma.

Lemma 8 *Let M be a popular matching. Then $M \cap E_1$ is a maximum matching of G_1.*

Next, work towards a generalized definition of $s(a)$. Restrict attention to rank-one edges, that is, to the graph G_1 and using M_1, partition $\mathcal{A} \cup \mathcal{P}$ into three disjoint sets. A node v is *even* (respectively *odd*) if there is an even (respectively odd) length alternating path (with respect to M_1) from an unmatched node to v. Similarly, a node v is *unreachable* if there is no alternating path (w.r.t. M_1) from an unmatched node to v. Denote by $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$ the sets of even, odd, and unreachable nodes, respectively. Conclude the following facts about $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$ by using the well-known Gallai–Edmonds decomposition theorem.

(a) $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$ are pairwise disjoint. Every maximum matching in G_1 partitions the vertex set into the same partition of even, odd, and unreachable nodes.

(b) In any maximum-cardinality matching of G_1, every node in $\mathcal{O}$ is matched with some node in $\mathcal{E}$, and every node in $\mathcal{U}$ is matched with another node in $\mathcal{U}$. The size of a maximum-cardinality matching is $|\mathcal{O}| + |\mathcal{U}|/2$.

(c) No maximum-cardinality matching of G_1 contains an edge between two nodes in $\mathcal{O}$, or a node in $\mathcal{O}$ and

a node in $\mathcal{U}$. And there is no edge in G_1 connecting a node in $\mathcal{E}$ with a node in $\mathcal{U}$.

The above facts motivate the following definition of $s(a)$: let $s(a)$ be the set of most preferred posts in a's preference list that are *even* in G_1 (note that $s(a) \neq \emptyset$, since $l(a)$ is always even in G_1). Recall that our original definition of $s(a)$ led to parts (2) and (3) of Lemma 5 which restrict the set of posts to which an applicant can be matched in a popular matching. This shows that the generalized definition leads to analogous results here.

Lemma 9 *Let M be a popular matching. Then for every applicant a, $M(a)$ can never be strictly between $f(a)$ and $s(a)$ on a's preference list and $M(a)$ can never be worse than $s(a)$ in a's preference list.*

The following characterization of popular matchings is formed.

Lemma 10 *A matching M is popular in G if and only if (i) $M \cap E_1$ is a maximum matching of G_1, and (ii) for each applicant a, $M(a) \in f(a) \cup s(a)$.*

Given an instance graph $G = (\mathcal{A} \cup \mathcal{P}, E)$, we define the *reduced graph* $G' = (\mathcal{A} \cup \mathcal{P}, E')$ as the subgraph of G containing edges from each applicant a to posts in $f(a) \cup s(a)$. The authors remark that G' need not admit an applicant-complete matching, since $l(a)$ is now isolated whenever $s(a) \neq \{l(a)\}$. Lemma 11 tells us that a matching is popular if and only if it belongs to the graph G' and it is a maximum matching on rank one edges. Recall that all popular matchings are applicant-complete through the introduction of last resorts. Hence, the following characterization is immediate.

Theorem 11 *M is a popular matching of G if and only if (i) $M \cap E_1$ is a maximum matching of G_1, and (ii) M is an applicant-complete matching of G'.*

Using the characterization in Theorem 11, the authors now present an efficient algorithm for solving the ranked matching problem.

Popular-Matching($G = (\mathcal{A} \cup \mathcal{P}, E)$)

1. Construct the graph $G' = (\mathcal{A} \cup \mathcal{P}, E')$, where $E' = \{(a, p) \mid p \in f(a) \cup s(a),\ a \in \mathcal{A}\}$.
2. Compute a maximum matching M_1 on rank one edges i. e., M_1 is a maximum matching in $G_1 = (\mathcal{A} \cup \mathcal{P}, E_1)$. ($M_1$ is also a matching in G' because $E' \supseteq E_1$)
3. Delete all edges in G' connecting two nodes in the set $\mathcal{O}$ or a node in $\mathcal{O}$ with a node in $\mathcal{U}$, where $\mathcal{O}$ and $\mathcal{U}$ are the sets of odd and unreachable nodes of $G_1 = (\mathcal{A} \cup \mathcal{P}, E_1)$.
   Determine a maximum matching M in the modified graph G' by augmenting M_1.
4. If M is not applicant-complete, then declare that there is no popular matching in G.
   Else return M.

The matching returned by the algorithm Popular-Matching is an applicant-complete matching in G' and it is a maximum matching on rank one edges. So the correctness of the algorithm follows from Theorem 11. It is easy to see that the running time of this algorithm is $O(\sqrt{n}m)$. The algorithm of Hopcroft and Karp [7] is uesd to compute a maximum matching in G_1 and identify the set of edges E' and construct G' in $O(\sqrt{n}m)$ time. Repeatedly augment M_1 (by the Hopcroft–Karp algorithm) to obtain M. This proves Theorem 3.

It is now a simple matter to solve the maximum-cardinality popular matching problem. Assume that the instance $G = (\mathcal{A} \cup \mathcal{P}, E)$ admits a popular matching. (Otherwise, the process is done.) In order to compute an applicant-complete matching in G' that is a maximum matching on rank one edges and which maximizes the number of applicants not matched to their last resort, first compute an arbitrary popular matching M' and remove all edges of the form $(a, l(a))$ from M' and from the graph G'. Call the resulting subgraph of G' as H. Determine a maximum matching N in H by augmenting M'. N need not be a popular matching, since it need not be a maximum matching in the graph G'. However, this is easy to mend. Determine a maximum matching M in G' by augmenting N. It is easy to show that M is a popular matching which maximizes the number of applicants not matched to their last resort. Since the algorithm takes $O(\sqrt{n}m)$ time, Theorem 4 is shown.

Applications

The bipartite matching problem with a graded edge set is well-studied in the economics literature, see for example [1,10,12]. It models some important real-world problems, including the allocation of graduates to training positions [8], and families to government-owned housing [11]. The concept of a popular matching was first introduced by Gardenfors [5] under the name *majority assignment* in the context of the stable marriage problem [4,6].

Various other definitions of optimality have been considered. For example, a matching is *Pareto-optimal* [1,2,10] if no applicant can improve his/her allocation (say by exchanging posts with another applicant) without requiring some other applicant to be worse off. Stronger defini-

tions exist: a matching is *rank-maximal* [9] if it allocates the maximum number of applicants to their first choice, and then subject to this, the maximum number to their second choice, and so on. A matching is *maximum utility* if it maximizes $\sum_{(a,p)\in M} u_{a,p}$, where $u_{a,p}$ is the utility of allocating post p to applicant a. Neither rank-maximal nor maximum-utility matchings are necessarily popular.

Cross References

- ▶ Hospitals/Residents Problem
- ▶ Maximum Matching
- ▶ Weighted Popular Matchings

Recommended Reading

1. Abdulkadiroğlu, A., Sönmez, T.: Random serial dictatorship and the core from random endowments in house allocation problems. Econom. **66**(3), 689–701 (1998)
2. Abraham, D.J., Cechlárová, K., Manlove, D.F., Mehlhorn, K.: Pareto-optimality in house al- location problems. In: Proceedings of the 15th International Symposium on Algorithms and Computation, (LNCS, vol. 3341), pp. 3–15. Springer, Sanya, Hainan (2004)
3. Abraham, D.J., Irving, R.W., Kavitha, T., Mehlhorn, K.: Popular matchings. In: Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms, pp. 424–432. SIAM, Vancouver (2005)
4. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Mon. **69**, 9–15 (1962)
5. Gardenfors, P.: Match Making: assignments based on bilateral preferences. Behav. Sci. **20**, 166–173 (1975)
6. Guseld, D., Irving, R.W.: The Stable Marriage Problem: Structure and Algorithms. MIT Press, Cambridge (1989)
7. Hopcroft, J.E., Karp, R.M.: A $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. SIAM J. Comput. **2**, 225–231 (1973)
8. Hylland, A., Zeckhauser, R.: The ecient allocation of individuals to positions. J. Political Econ. **87**(2), 293–314 (1979)
9. Irving, R.W., Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.: Rank-maximal matchings. In: Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms, pp. 68–75. SIAM, New Orleans (2004)
10. Roth, A.E., Postlewaite, A.: Weak versus strong domination in a market with indivisible goods. J. Math. Econ. **4**, 131–137 (1977)
11. Yuan, Y.: Residence exchange wanted: a stable residence exchange problem. Eur. J. Oper. Res. **90**, 536–546 (1996)
12. Zhou, L.: On a conjecture by Gale about one-sided matching problems. J. Econ. Theory **52**(1), 123–135 (1990)

Rank and Select Operations on Binary Strings

1974; Elias

Naila Rahman, Rajeev Raman
Department of Computer Science,
University of Leicester, Leicester, UK

Keywords and Synonyms

Binary bit-vector; Compressed bit-vector, Rank and select dictionary; Fully indexable dictionary (FID)

Problem Definition

Given a static sequence $\boldsymbol{b} = b_1 \ldots b_m$ of m bits, to preprocess the sequence and to create a space-efficient data structure that supports the following operations rapidly:

$\mathsf{rank}_1(i)$ takes an index i as input, $1 \leq i \leq m$, and returns the number of **1**s among $b_1 \ldots b_i$.

$\mathsf{select}_1(i)$ takes an index $i \geq 1$ as input, and returns the position of the ith **1** in $\boldsymbol{b}$, and -1 if i is greater than the number of **1**s in $\boldsymbol{b}$.

The operations rank_0 and select_0 are defined analogously for the **0**s in $\boldsymbol{b}$. As $\mathsf{rank}_0(i) = i - \mathsf{rank}_1(i)$, one considers just rank_1 (abbreviated to rank), and refers to select_0 and select_1 collectively as select. In what follows, $|\boldsymbol{x}|$ denotes the length of a bit sequence $\boldsymbol{x}$ and $w(\boldsymbol{x})$ denotes the number of **1**s in it. $\boldsymbol{b}$ is always used to denote the input bit sequence, m to denote $|\boldsymbol{b}|$ and n to denote $w(\boldsymbol{b})$.

Models of Computation, Time and Space Bounds

Two models of computation are commonly considered. One is the *unit-cost RAM* model with word size $O(\lg m)$ bits [1]. The other model, which is particularly useful for proving lower bounds, is the *bit-probe* model, where the data structure is stored in bit-addressable memory, and the complexity of answering a query is the worst-case number of bits of the data structure that are probed by the algorithm to answer that query. In the RAM model, the algorithm can read $O(\lg m)$ *consecutive* bits in one step, so supporting all operations in $O(1)$ time on the RAM model implies a solution that uses $O(\lg m)$ bit-probes, but the converse is not true.

This entry considers three variants of the problem: in each variant, rank and select must be supported in $O(1)$ time on the RAM model, or in $O(\lg m)$ bit-probes. However, the use of memory varies:

Problem 1 (Bit-Vector) *The overall space used must be $m + o(m)$ bits.*

Problem 2 (Bit-Vector Index) *$\boldsymbol{b}$ is given in read-only memory and the algorithm can create auxiliary data structures (called indices) which must use $o(m)$ bits.*

Indices allow the representation of $\boldsymbol{b}$ to be de-coupled from the auxiliary data structure, e. g., $\boldsymbol{b}$ can be stored (in a potentially highly compressed form) in a data structure

such as that of [6,9,17] which allows access to $O(\lg m)$ consecutive bits of $\boldsymbol{b}$ in $O(1)$ time on the RAM model. Most bit-vectors developed to date are bit-vector indices.

Recalling that $n = w(\boldsymbol{b})$, observe that if m and n are known to an algorithm, there are only $l = \binom{m}{n}$ possibilities for $\boldsymbol{b}$, so an information-theoretically optimal encoding of $\boldsymbol{b}$ would require $B(m, n) = \lceil \lg l \rceil$ bits (it can be verified that $B(m, n) < m$ for all m, n). The next problem is:

Problem 3 (Compressed Bit-Vector) *The overall space used must be $B(m, n) + o(n)$ bits.*

It is helpful to understand the asymptotics of $B(m, n)$ in order to appreciate the difference between the bit-vector and the compressed bit-vector problems:

- Using standard approximations of the factorial function, one can show [14] that:

$$B(m, n) = n \lg(m/n) + n \lg e + O(n^2/m) \quad (1)$$

  If $n = o(m)$, then $B(m, n) = o(m)$, and if such a sparse sequence $\boldsymbol{b}$ were represented as a compressed bit-vector, then it would occupy $o(m)$ bits, rather than $m + o(m)$ bits.
- $B(m, n) = m - O(\lg m)$, whenever $|m/2 - n| = O(\sqrt{m \lg m})$. In such cases, a compressed bit-vector will take about the same amount of space as a bit-vector.
- Taking $p = n/m, H_0(\boldsymbol{b}) = (1/p)\lg(1/p) + (1/(1-p))\lg(1/(1-p))$ is the *empirical zeroth-order entropy* of $\boldsymbol{b}$. If $\boldsymbol{b}$ is compressed using an 'entropy' compressor such as non-adaptive arithmetic coding [18], the size of the compressed output is at least $mH_0(\boldsymbol{b})$ bits. However, $B(m, n) = mH_0(\boldsymbol{b}) - O(\log m)$. Applying Golomb coding to the 'gaps' between successive **1**s, which is the best way to compress bit sequences that represent inverted lists [18], also gives a space usage close to $B(m, n)$ [4].

Related Problems

Viewing $\boldsymbol{b}$ as the characteristic vector of a set $S \subseteq U = \{1, \ldots, m\}$, note that the well-known *predecessor* problem – given $y \in U$, return $pred(y) = \max\{z \in S | z \leq y\}$ – may be implemented as $\mathsf{select}_1(\mathsf{rank}_1(y))$. One may also view $\boldsymbol{b}$ as a *multiset* of size $m - n$ over the universe $\{1, \ldots, n+1\}$ [5]. First, append a **1** to $\boldsymbol{b}$. Then, take each of the $n+1$ **1**s to be the elements of the universe, and the number of consecutive **0**s immediately preceding a **1** to indicate their multiplicities. For example, $\boldsymbol{b}$ = **01100100** maps to the multiset $\{1, 3, 3, 4, 4\}$. Seen this way, $\mathsf{select}_1(i) - i$ on $\boldsymbol{b}$ gives the number of items in the multiset that are $\leq i$, and $\mathsf{select}_0(i) - i + 1$ gives the value of the ith element of the multiset.

Lower-Order Terms

From an asymptotic viewpoint, the space utilization is dominated by the main terms in the space bound. However, the second (apparently lower-order) terms are of interest for several reasons, primarily because the lower-order terms are extremely significant in determining practical space usage, and also because non-trivial space bounds have been proven for the size of the lower-order terms.

Key Results

Reductions

It has been already noted that rank_0 and rank_1 reduce to each other, and that operations on multisets reduce to select operations on a bit sequence. Some other reductions, whereby one can support operations on $\boldsymbol{b}$ by performing operations on bit sequences derived from $\boldsymbol{b}$ are:

Theorem 1

(a) rank *reduces to* select_0 *on a bit sequence* $\boldsymbol{c}$ *such that* $|\boldsymbol{c}| = m + n$ *and* $w(\boldsymbol{c}) = n$.

(b) *If* $\boldsymbol{b}$ *has no consecutive* **1**s, *then* select_0 *on* $\boldsymbol{b}$ *can be reduced to* rank *on a bit sequence* $\boldsymbol{c}$ *such that* $|\boldsymbol{c}| = m - n$ *and* $w(\boldsymbol{c})$ *is either* $n - 1$ *or* n.

(c) *From* $\boldsymbol{b}$ *one can derive two bit sequences* $\boldsymbol{b_0}$ *and* $\boldsymbol{b_1}$ *such that* $|\boldsymbol{b_0}| = m - n, |\boldsymbol{b_1}| = n, w(\boldsymbol{b_0}), w(\boldsymbol{b_1}) \leq \min\{m - n, n\}$ *and* select_0 *and* select_1 *on* $\boldsymbol{b}$ *can be supported by supporting* select_1 *and* rank *on* b_0 *and* b_1.

Parts (a) and (b) follow from Elias's observations on multiset representations (see the "Related Problems" paragraph), specialized to sets. For part (a), create $\boldsymbol{c}$ from $\boldsymbol{b}$ by adding a **0** after every **1**. For example, if $\boldsymbol{b}$ = **01100100** then $\boldsymbol{c}$ = **010100010000**. Then, $\mathsf{rank}_1(i)$ on $\boldsymbol{b}$ equals $\mathsf{select}_0(i) - i$ on $\boldsymbol{c}$. For part (b), essentially invert the mapping of part (a). Part (c) is shown in [3].

Bit-Vector Indices

Theorem 2 ([8]) *There is an index of size* $(1 + o(1))(m \lg \lg m/\lg m) + O(m/\lg m)$ *that supports* rank *and* select *in O(1) time on the RAM model.*

Elias previously gave an $o(m)$-bit index that supported select in $O(\lg m)$ bit-probes on average (where the average was computed across all select queries). Jacobson gave $o(m)$-bit indices that supported rank and select in $O(\lg m)$ bit-probes in the worst case. Clark and Munro [2] gave the first $o(m)$-bit indices that support both rank and select in $O(1)$ time on the RAM. A matching lower bound on the

size of indices has also been shown (this also applies to indices which support rank and select in $O(1)$ time on the RAM model):

Theorem 3 ([8]) *Any index that allows* rank *or* select_1 *to be supported in* $O(\lg m)$ *bit-probes has size* $\Omega(m \lg\lg m/\lg m)$ *bits.*

Compressed Bit-Vectors

Theorem 4 *There is a compressed bit-vector that uses:*

- ***(a)*** $B(m,n) + O(m \lg\lg m/\lg m)$ *bits and supports* rank *and* select *in* $O(1)$ *time.*
- ***(b)*** $B(m,n) + O(n(\lg\lg n)^2/\lg n)$ *bits and supports* rank *in* $O(1)$ *time, when* $n = m/(\lg m)^{O(1)}$.
- ***(c)*** $B(m,n) + O(n \lg\lg n/\sqrt{\lg n})$ *bits and supports* select_1 *in* $O(1)$ *time.*

Theorem 4(a) and (c) were shown by Raman et al. [16] and Theorem 4(b) by Pagh [14]. Note that Theorem 4(a) has a lower-order term that is $o(m)$, rather than $o(n)$ as required by the problem statement. As compressed bit-vectors must represent $\boldsymbol{b}$ compactly, they are not bit-vector indices, and the lower bound of Theorem 3 does not apply to compressed bit-vectors. Coarser lower bounds are obtained by reduction to the predecessor problem on sets of integers, for which tight upper and lower bounds in the RAM model are now known. In particular the work of [15] implies:

Theorem 5 *Let* $U = \{1,\ldots,M\}$ *and let* $S \subseteq U, |S| = N$. *Any data structure on a RAM with word size* $O(\lg M)$ *bits that occupies at most* $O(N \lg M)$ *bits of space can support predecessor queries on* S *in* $O(1)$ *time only when* $N = M/(\lg M)^{O(1)}$ *or* $N = (\lg M)^{O(1)}$.

As noted in the paragraph "Related Problems", the predecessor problem can be solved by the use of rank and select_1 operations. Thus, Theorem 5 has consequences for compressed bit-vector data structures, which are spelt out below:

Corollary 1 *There is no data structure that uses* $B(m,n) + o(n)$ *bits and supports either* rank *or* select_0 *in* $O(1)$ *time unless* $n = m/(\lg m)^{O(1)}$, *or* $n = (\log m)^{O(1)}$.

Given a set $S \subseteq U = \{1,\ldots,m\}, |S| = n$, we have already noted that the predecessor problem on S is equivalent to rank and select_1 on a bit-vector $\boldsymbol{c}$ with $w(\boldsymbol{c}) = n$, and $|\boldsymbol{c}| = m$. However, $B(m,n) + o(n) = O(n \lg m)$. Thus, given a bit-vector that uses $B(m,n) + o(n)$ bits and supports rank in $O(1)$ time for $m = n(\lg n)^{\omega(1)}$, we can augment it with the trivial $O(1)$-time data structure for select_1, that stores the value of $\mathsf{select}_1(i)$ for $i = 1,\ldots,n$ (which occupies a further $O(n \lg m)$ bits), solving the predecessor problem in $O(1)$ time, a contradiction. The hardness of select_0 is shown in [16], but follows easily from Theorem 1(a) and Eq. (1).

Applications

There are a vast number of applications of bit-vectors in succinct and compressed data structures (see e. g. [12]). Such data structures are used for, e. g., text indexing, compact representations of graphs and trees, and representations of semi-structured (XML) data.

Experimental Results

Several teams have implemented bit-vectors and compressed bit-vectors. When implementing bit-vectors for good practical performance, both in terms of speed and space usage, the lower-order terms are very important, even for uncompressed bit-vectors[1], and can dominate the space usage even for bit-vector sizes that are at the limit of conceivable future practical interest. Unfortunately, this problem may not be best addressed purely by a theoretical analysis of the lower-order terms. Bit-vectors work by partitioning the input bit sequence into (usually equal-sized) blocks at several levels of granularity – usually 2–3 levels are needed to obtain a space bound of $m + o(m)$ bits. However, better space usage – as well as better speed – in practice can be obtained by reducing the number of levels, resulting in space bounds of the form $(1+\epsilon)m$ bits, for any $\epsilon > 0$, with support for rank and select in $O(1/\epsilon)$ time.

Clark [2] implemented bit-vectors for external-memory suffix trees. More recently, an implementation using ideas of Clark and Jacobson was used by [7], which occupied $(1+\epsilon)m$ bits and supported operations in $O(1/\epsilon)$ time. Using a substantially different approach, Kim et al. [11] gave a bit-vector that takes $(2+\epsilon)n$ bits to support rank and select. Experiments using bit sequences derived from real-world data in [3,4] showed that if parameters are set to ensure that [11] and [7] use similar space – on *typical* inputs – the Clark–Jacobson implementation of [7] is somewhat faster than an implementation of [11]. On some inputs, the Clark–Jacobson implementation can use significantly more space, whereas Kim et al.'s bit-vector appears to have stable space usage; Kim et al.'s bit-vector may also be superior for somewhat sparse bit-vectors. Combining ideas from [7,11], a third practical bit-vector (which is not

[1] For compressed bit-vectors, the 'lower-order' $o(m)$ or $o(n)$ term can dominate $B(m,n)$, but this is not our concern here.

a bit-vector index) was described in [4], and appears to have desirable features of both [11] and [7]. A first implementational study on compressed bit-vectors can be found in [13] (compressed bit-vectors supporting only select_1 were considered in [4]).

URL to Code

Bit-vector implementations from [3,4,7] can be found at http://hdl.handle.net/2381/318.

Cross References

- ► Arithmetic Coding for Data Compression
- ► Compressed Text Indexing
- ► Succinct Encoding of Permutations: Applications to Text Indexing
- ► Tree Compression and Indexing

Recommended Reading

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley (1974)
2. Clark, D., Munro, J.I.: Efficient suffix trees on secondary storage. In: Proc. 7th ACM-SIAM SODA, pp. 383–391 (1996)
3. Delpratt, O., Rahman, N., Raman, R.: Engineering the LOUDS succinct tree representation. In: Proc. WEA 2006. LNCS, vol. 4007, pp. 134–145. Springer, Berlin (2006)
4. Delpratt, O., Rahman, N., Raman, R.: Compressed prefix sums. In: Proc. SOFSEM 2007. LNCS, vol. 4362, pp. 235–247 (2007)
5. Elias, P.: Efficient storage retrieval by content and address of static files. J. ACM, **21**(2):246–260 (1974)
6. Ferragina, P., Venturini, R.: A simple storage scheme for strings achieving entropy bounds. Theor. Comput. Sci. **372**, 115–121 (2007)
7. Geary, R.F., Rahman, N., Raman, R., Raman, V.: A simple optimal representation for balanced parentheses. Theor. Comput. Sci. **368**, 231–246 (2006)
8. Golynski, A.: Optimal lower bounds for rank and select indexes. In: Proc. ICALP 2006, Part I. LNCS, vol. 4051, pp. 370–381 (2006)
9. González, R., Navarro, G.: Statistical encoding of succinct data structures. In: Proc. CPM 2006. LNCS, vol. 4009, pp. 294–305. Springer, Berlin (2006)
10. Jacobson, G.: Space-efficient static trees and graphs. In: Proc. 30th FOCS, pp. 549–554 (1989)
11. Kim, D.K., Na, J.C., Kim, J.E., Park, K.: Efficient implementation of Rank and Select functions for succinct representation. In: Proc. WEA 2005. LNCS, vol. 3505, pp. 315–327 (2005)
12. Munro, J.I., Srinivasa Rao, S.: Succinct representation of data structures. In: Mehta, D., Sahni, S. (eds.) Handbook of Data Structures with Applications, Chap 37. Chapman and Hall/CRC Press (2005)
13. Okanohara, D., Sadakane, K.: Practical entropy-compressed rank/select dictionary. In: Proc. 9th ACM-SIAM Workshop on Algorithm Engineering and Experiments (ALENEX '07), SIAM, to appear (2007)
14. Pagh, R.: Low redundancy in static dictionaries with constant query time. SIAM J. Comput. **31**, 353–363 (2001)
15. Patrascu, M., Thorup, M.: Time-space trade-offs for predecessor search. In: Proc. 38th ACM STOC, pp. 232–240 (2006)
16. Raman, R., Raman, V., Rao, S.S.: Succinct indexable dictionaries, with applications to representing k-ary trees and multisets. In: Proc. 13th ACM-SIAM SODA, pp. 233–242 (2002)
17. Sadakane, K., Grossi, R.: Squeezing succinct data structures into entropy bounds. In: Proc. 17th ACM-SIAM SODA, pp. 1230–1239. ACM Press (2006)
18. Witten, I., Moffat, A., Bell, I.: Managing Gigabytes, 2nd edn. Morgan Kaufmann (1999)

Rate Adjustment and Allocation

► Schedulers for Optimistic Rate Based Flow Control

Rate-Monotonic Scheduling

1973; Liu, Layland

NATHAN FISHER, SANJOY BARUAH
Department of Computer Science,
University of North Carolina, Chapel Hill, NC, USA

Keywords and Synonyms

Real-time systems; Static-priority scheduling; Fixed-priority scheduling; Rate-monotonic analysis

Problem Definition

Liu and Layland [9] introduced rate-monotonic scheduling in the context of the scheduling of recurrent real-time processes upon a computing platform comprised of a single preemptive processor.

The Periodic Task Model

The *periodic task* abstraction models real-time processes that make repeated requests for computation. As defined in [9], each periodic task τ_i is characterized by an ordered pair of positive real-valued parameters (C_i, T_i), where C_i is the *worst-case execution requirement* and T_i the *period* of the task. The requests for computation that are made by task τ_i (subsequently referred to as *jobs* that are *generated* by τ_i) satisfy the following assumptions:

A1: τ_i's first job arrives at system start time (assumed to equal time zero), and subsequent jobs arrive every T_i time units. I.e., one job arrives at time-instant $k \times T_i$ for all integer $k \geq 0$.

A2: Each job needs to execute for at most C_i time units. I.e., C_i is the maximum amount of time that a processor would require to execute each job of τ_i, without interruption.

A3: Each job of τ_i must complete before the next job arrives. That is, each job of task τ_i must complete execution by a *deadline* that is T_i time-units after its arrival time.

A4: Each task is *independent* of all other tasks – the execution of any job of task τ_i is not contingent on the arrival or completion of jobs of any other task τ_j.

A5: A job of τ_i may be *preempted* on the processor without additional execution cost. In other words, if a job of τ_i is currently executing, then it is permitted that this execution be halted, and a job of a different task τ_j begin execution immediately.

A periodic task system $\tau \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \ldots, \tau_n\}$ is a collection of n periodic tasks. The *utilization* $U(\tau)$ is defined as follows:

$$U(\tau) \stackrel{\text{def}}{=} \sum_{i=1}^{n} C_i/T_i \ . \tag{1}$$

Intuitively, this denotes the fraction of time that may be spent by the processor executing jobs of tasks in τ, in the worst case.

The Rate-Monotonic Scheduling Algorithm

A (uniprocessor) scheduling algorithm determines which task executes on the shared processor at each time-instant. If a scheduling algorithm is guaranteed to always meet all deadlines when scheduling a task system τ, then τ is said to be *schedulable* with respect to that scheduling algorithm.

Many scheduling algorithms work as follows: At each time-instant, they assign a priority to each job, and select for execution the greatest-priority job with remaining execution. A *static priority* (often called *fixed priority*) scheduling algorithm for scheduling periodic tasks is one in which it is required that all the jobs of each periodic task be assigned the same priority.

Liu and Layland [9] proposed the *rate-monotonic* (RM) static priority scheduling algorithm, which assigns priority to jobs according to the period parameter of the task that generates them: *the smaller the period, the higher the priority*. Hence if $T_i < T_j$ for two tasks τ_i and τ_j, then each job of τ_i has higher priority than all jobs of τ_j and hence any executing job of τ_j will be preempted by the arrival of one of τ_i's jobs. Ties may be broken arbitrarily but consistently – if $T_i = T_j$, then either all jobs of τ_i are assigned higher priority than all jobs of τ_j, or all jobs of τ_j are assigned higher priority than all jobs of τ_i.

Key Results

Results from the original paper by Liu and Layland [9] are presented in Sect. "Results from [9]" below; results extending the original work are briefly described in Sect. "Results since [9]".

Results from [9]

Optimality Liu and Layland were concerned with designing "good" static priority scheduling algorithms. They defined a notion of optimality for such algorithms: A static priority algorithm $\mathcal{A}$ is *optimal* if any periodic task system that is schedulable with respect to some static priority algorithm is also schedulable with respect to $\mathcal{A}$.

Liu and Layland obtained the following result for the rate-monotonic scheduling algorithm (RM):

Theorem 1 *For periodic task systems,* RM *is an optimal static priority scheduling algorithm.*

Schedulability Testing A *schedulability test* for a particular scheduling algorithm determines, for any periodic task system τ, whether τ is schedulable with respect to that scheduling algorithm. A schedulability test is said to be *exact* if it is the case that it correctly identifies all schedulable task systems, and *sufficient* if it identifies some, but not necessarily all, schedulable task systems.

In order to derive good schedulability tests for the rate-monotonic scheduling algorithm, Liu and Layland considered the concept of *response time*. The response time of a job is defined as the elapsed time between the arrival of a job and its completion time in a schedule; the response time of a task is defined to be the largest response time that may be experienced by one its jobs. For static priority scheduling, Liu and Layland obtained the following result on the response time:

Theorem 2 *The maximum response time for a periodic task τ_i occurs when a job of τ_i arrives simultaneously with jobs of all higher-priority tasks. Such a time-instant is known as the critical instant for task τ_i.*

Observe that the critical instant of the lowest-priority task in a periodic task system is also a critical instant for all tasks of higher priority. An immediate consequence of the previous theorem is that the response-time of each task in the periodic task system can be obtained by simulating the scheduling of the periodic task system starting at the critical instant of the lowest-priority task. If the response time for each task τ_i obtained from such simulation does not exceed T_i, then the task system will always meet all deadlines when scheduled according to the given

priority assignment. This argument immediately gives rise to a schedulability analysis test [7] for any static priority scheduling algorithm. Since the simulation may need to be carried out until $\max_{i=1}^{n}\{T_i\}$, this schedulability test has run-time pseudo-polynomial in the representation of the task system:

Theorem 3 ([7]) *Exact rate-monotonic schedulability testing of a periodic task system may be done in time pseudo-polynomial in the representation in the task system.*

Liu and Layland also derived a polynomial-time sufficient (albeit not exact) schedulability test for RM, based upon the utilization of the task system:

Theorem 4 *Let n denote the number of tasks in periodic task system τ. If $U(\tau) \leq n(2^{1/n} - 1)$, then τ is schedulable with respect to the* RM *scheduling algorithm.*

Results since [9]

The utilization-bound sufficient schedulability test (Theorem 4) was shown to be tight in the sense that for all n, there are unschedulable task systems comprised of n tasks with utilization exceeding $n(2^{1/n} - 1)$ by an arbitrarily small amount. However, tests have been devised that exploit more knowledge about tasks' period parameters. For instance, Kuo and Mok [6] provide a potentially superior utilization bound for task systems in which the task period parameters tend to be harmonically related – exact multiples of one another. Suppose that a collection of numbers is said to comprise a *harmonic chain* if for every two numbers in the set, it is the case that one is an exact multiple of the other. Let $\tilde{n}$ denote the minimum number of harmonic chains into which the period parameters $\{T_i\}_{i=1}^{n}$ of tasks in τ may be partitioned; a sufficient condition for task system τ to be RM-schedulable is that

$$U(\tau) \leq \tilde{n}(2^{1/\tilde{n}} - 1).$$

Since $\tilde{n} \leq n$ for all task systems τ, this utilization bound above is never inferior to the one in Theorem 4, and is superior for all τ for which $\tilde{n} < n$.

A different polynomial-time schedulability test was proposed by Bini, Buttazzo, and Buttazzo [3]: they showed that

$$\prod_{i=1}^{n}((C_i/T_i) + 1) \leq 2$$

is sufficient to guarantee that the periodic task system $\{\tau_1, \tau_2, \ldots, \tau_n\}$ is rate-monotonic schedulable. This test is commonly referred to as the *hyperbolic* schedulability test for rate-monotonic schedulability. The hyperbolic test is in general known to be superior to the utilization-based test of Theorem 4 – see [3] for details.

Other work done since the seminal paper of Liu and Layland has focused on relaxing the assumptions of the periodic task model.

The (implicit-deadline) *sporadic* task model relaxed assumption A1 by allowing T_i to be the *minimum* (rather than exact) separation between arrivals of successive jobs of task τ_i. It turns out that the results in Sect. "Results from [9]" – Theorems 1–4 – hold for systems of such tasks as well.

A more general sporadic task model has also been studied that relaxes assumption A3 in addition to assumption A1, by allowing for the explicit specification of a deadline parameter for each task (which may differ from the task's period). The *deadline monotonic* scheduling algorithm [8] generalizes rate-monotonic scheduling to such task systems.

Work has also been done [2,10] in removing the independence assumption of A4, by allowing for different tasks to use critical sections to access non-preemptable serially reusable resources.

Current work is focused on scheduling tasks on multiprocessor or distributed systems where one or more of the assumptions A1–A5 have been relaxed. In addition, recent work has relaxed the assumption (A2) that worst-case execution requirement is known and instead probabilistic execution requirement distributions are considered [4].

Applications

The periodic task model has been invaluable for modeling several different types of systems. For control systems, the periodic task model is well-suited for modeling the periodic requests and computations of sensors and actuators. Multimedia and network applications also typically involve computation of periodically arriving packets and data. Many operating systems for real-time systems provide support for periodic tasks as a standard primitive.

Many of the results described in Sect. "Key Results" above have been integrated into powerful tools, techniques, and methodologies for the design and analysis of real-time application systems [1,5]. Although these are centered around the deadline-monotonic rather than rate-monotonic scheduling algorithm, the general methodology is commonly referred to as the *rate-monotonic analysis* (RMA) methodology.

Open Problems

There are plenty of interesting and challenging open problems in real-time scheduling theory; however, most of

these are concerned with extensions to the basic task and scheduling model considered in the original Liu and Layland paper [9]. Perhaps the most interesting open problem with respect to the task model in [9] is regarding the computational complexity of schedulability analysis of static priority scheduling. While all known exact tests (e. g., Theorem 3) run in pseudo-polynomial time and all known polynomial-time tests are sufficient rather than exact, there has been no significant result pigeonholing the computational complexity of static priority schedulability analysis for periodic task systems.

Cross References

- ▶ List Scheduling
- ▶ Load Balancing
- ▶ Schedulers for Optimistic Rate Based Flow Control
- ▶ Shortest Elapsed Time First Scheduling

Recommended Reading

1. Audsley, N., Burns, A., Wellings, A.: Deadline monotonic scheduling theory and application. Control Eng. Pract. **1**, 71–78 (1993)
2. Baker, T.P.: Stack-based scheduling of real-time processes. Real-Time Systems: The Int. J. Time-Critical Comput. **3**, 67–100 (1991)
3. Bini, E., Buttazzo, G., Buttazzo, G.: Rate monotonic scheduling: The hyperbolic bound. IEEE Trans. Comput. **52**, 933–942 (2003)
4. Gardener, M.K.: Probabilistic Analysis and Scheduling of Critical Soft Real-Time Systems. Ph. D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign (1999)
5. Klein, M., Ralya, T., Pollak, B., Obenza, R., Harbour, M.G.: A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems. Kluwer Academic Publishers, Boston (1993)
6. Kuo, T.-W., Mok, A.K.: Load adjustment in adaptive real-time systems. In: Proceedings of the IEEE Real-Time Systems Symposium, pp. 160–171. San Antonio, December 1991
7. Lehoczky, J., Sha, L., Ding, Y.: The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In: Proceedings of the Real-Time Systems Symposium – 1989, Santa Monica, December 1989. IEEE Computer Society Press, pp. 166–171
8. Leung, J., Whitehead, J.: On the complexity of fixed-priority scheduling of periodic, real-time tasks. Perform. Eval. **2**, 237–250 (1982)
9. Liu, C., Layland, J.: Scheduling algorithms for multiprogramming in a hard real-time environment. J. ACM **20**, 46–61 (1973)
10. Rajkumar, R.: Synchronization In Real-Time Systems – A Priority Inheritance Approach. Kluwer Academic Publishers, Boston (1991)

Real-Time Systems

- ▶ Rate-Monotonic Scheduling

Rectilinear Spanning Tree

2002; Zhou, Shenoy, Nicholls

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

Keywords and Synonyms

Metric minimum spanning tree; Rectilinear spanning graph

Problem Definition

Given a set of n points in a plane, a spanning tree is a set of edges that connects all the points and contains no cycles. When each edge is weighted using some distance metric of the incident points, the *metric minimum spanning tree* is a tree whose sum of edge weights is minimum. If the Euclidean distance (L_2) is used, it is called the *Euclidean minimum spanning tree*; if the rectilinear distance (L_1) is used, it is called the *rectilinear minimum spanning tree*.

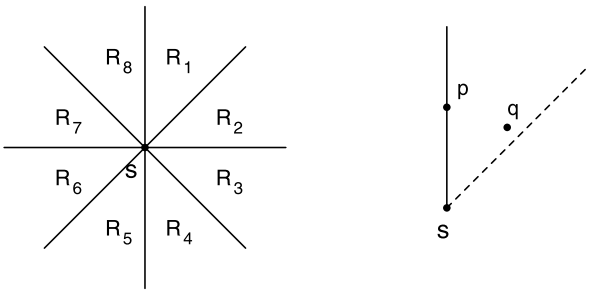
Since the minimum spanning tree problem on a weighted graph is well studied, the usual approach for metric minimum spanning tree is to first define an weighted graph on the set of points and then to construct a spanning tree on it.

Much like a connection graph is defined for the maze search [4], a spanning graph can be defined for the minimum spanning tree construction.

Definition 1 Given a set of points V in a plane, an undirected graph $G = (V, E)$ is called a *spanning graph* if it contains a minimum spanning tree of V in the plane.

Since spanning graphs with fewer edges give more efficient minimum spanning tree construction, the *cardinality* of a spanning graph is defined as its number of edges. It is easy to see that a complete graph on a set of points contains all spanning trees, thus is a spanning graph. However, such a graph has a cardinality of $O(n^2)$. A rectilinear spanning graph of cardinality $O(n)$ can be constructed within $O(n \log n)$ time [6] and will be described here.

Minimum spanning tree algorithms usually use two properties to infer the inclusion and exclusion of edges in a minimum spanning tree. The first property is known as the *cut property*. It states that an edge of smallest weight crossing any partition of the vertex set into two parts belongs to a minimum spanning tree. The second property is known as the *cycle property*. It says that an edge with largest weight in any cycle in the graph can be safely deleted. Since the two properties are stated in connection



Rectilinear Spanning Tree, Figure 1
Octal partition and the uniqueness property

with the construction of a minimum spanning tree, they are useful for a spanning graph.

Key Results

Using the terminology given in [3], the *uniqueness property* is defined as follows.

Definition 2 Given a point s, a region R has the *uniqueness property* with respect to s if for every pair of points $p, q \in R$, $||pq|| < \max(||sp||, ||sq||)$. A partition of space into a finite set of disjoint regions is said to have the uniqueness property with respect to s if each of its regions has the uniqueness property with respect to s.

The notation $||sp||$ is used to represent the distance between s and p under the L_1 metric. Define the *octal partition* of the plane with respect to s as the partition induced by the two rectilinear lines and the two 45 degree lines through s, as shown in Fig. 2a. Here, each of the regions R_1 through R_8 includes only one of its two bounding half line as shown in Fig. 2b. It can be shown that the octal partition has the uniqueness property.

Lemma 1 *Given a point s in the plane, the octal partition with respect to s has the uniqueness property.*

Proof To show a partition has the uniqueness property, it needs to prove that each region of the partition has the uniqueness property. Since the regions R_1 through R_8 are similar to each other, a proof for R_1 will be sufficient.

The points in R_1 can be characterized by the following inequalities

$$x \geq x_s\ ,$$
$$x - y < x_s - y_s\ .$$

Suppose there are two points p and q in R_1. Without loss of generality, it can be assumed $x_p \leq x_q$. If $y_p \leq y_q$, then $||sq|| = ||sp|| + ||pq|| > ||pq||$. Therefore it only needs to consider the case when $y_p > y_q$. In this case,

$$\begin{aligned} ||pq|| &= |x_p - x_q| + |y_p - y_q| \\ &= x_q - x_p + y_p - y_q \\ &= (x_q - y_q) + y_p - x_p \\ &< (x_s - y_s) + y_p - x_s \\ &= y_p - y_s \\ &\leq x_p - x_s + y_p - y_s \\ &= ||sp||\ . \end{aligned}$$

□

Given two points p, q in the same octal region of point s, the uniqueness property says that $||pq|| < \max(||sp||, ||sq||)$. Consider the cycle on points s, p, and q. Based on the cycle property, only one point with the minimum distance from s needs to be connected to s. An interesting property of the octal partition is that the contour of equi-distant points from s forms a line segment in each region. In regions R_1, R_2, R_5, R_6, these segments are captured by an equation of the form $x + y = c$; in regions R_3, R_4, R_7, R_8, they are described by the form $x - y = c$.

From each point s, the closest neighbor in each octant needs to be found. It will be described how to efficiently compute the neighbors in R_1 for all points. The case for other octant is symmetric. For the R_1 octant, a sweep line algorithm will run on all points according to non-decreasing $x + y$. During the sweep, maintained will be an *active* set consisting of points whose nearest neighbors in R_1 are yet to be discovered. When a point p is processed, all points in the active set that have p in their R_1 regions will be found. If s is such a point in the active set, since points are scanned in non-decreasing $x + y$, then p must be the nearest point in R_1 for s. Therefore, the edge sp will be added and s will be deleted from the active set. After processing those active points, the point p will be added into the active set. Each point will be added and deleted at most once from the active set.

A fundamental operation in the sweep line algorithm is to find a subset of active points such that a given point p is in their R_1 regions. Based on the observation that point p is in the R_1 region of point s if and only if s is in the R_5 region of p, it needs to find the subset of active points in the R_5 region of p. Since R_5 can be represented as a two-dimensional range $(-\infty, x_p] \times (x_p - y_p, +\infty)$ on $(x, x - y)$, a priority search tree [1] can be used to maintain the active point set. Since each of the insertion and deletion operations takes $O(\log n)$ time, and the query operation takes $O(\log n + k)$ time where k is the number of objects within the range, the total time for the sweep is

$O(n \log n)$. Since other regions can be processed in the similar way as in R_1, the algorithm is running in $O(n \log n)$ time. Priority search tree is a data structure that relies on maintaining a balanced structure for the fast query time. This works well for static input sets. When the input set is dynamic, re-balancing the tree can be quite challenging. Fortunately, the active set has a structure that can be explored for an alternate representation. Since a point is deleted from the active set if a point in its R_1 region is found, no point in the active set can be in the R_1 region of another point in the set.

Lemma 2 *For any two points p, q in the active set, it must be* $x_p \neq x_q$, *and if* $x_p < x_q$ *then* $x_p - y_p \leq x_q - y_q$.

Based on this property, the active set can be ordered in increasing order of x. This implies a non-decreasing order on $x - y$. Given a point s, the points which have s in their R_1 region must obey the following inequalities

$$x \leq x_s \,,$$
$$x - y > x_s - y_s \,.$$

To find the subset of active points which have s in their R_1 regions, it can first find the largest x such that $x \leq x_s$, then proceed in decreasing order of x until $x - y \geq x_s - y_s$. Since the ordering is kept on only one dimension, using any binary search tree with $O(\log n)$ insertion, deletion, and query time will also give us an $O(n \log n)$ time algorithm. Binary search trees also need to be balanced. An alternative is to use skip-lists [2] which use randomization to avoid the problem of explicit balancing but provide $O(\log n)$ expected behavior.

A careful study also shows that after the sweep process for R_1, there is no need to do the sweep for R_5, since all edges needed in that phase are either connected or implied. Moreover, based on the information in R_5, the number of edge connections can be further reduced. When the sweep step processes point s, it finds a subset of active points which have s in their R_1 regions. Without lost of generality, suppose p and q are two of them. Then p and q are in the R_5 region of s, which means $||pq|| < \max(||sp||, ||sq||)$. Therefore, it needs only to connect s with the nearest active point.

Since R_1 and R_2 have the same sweep sequence, they can be processed together in one pass. Similarly, R_3 and R_4 can be processed together in another pass. Based on the above discussion, the pseudo-code of the algorithm is presented in Fig. 2.

The correctness of the algorithm is stated in the following theorem.

Rectilinear Spanning Graph Algorithm

```
for (i = 0; i < 2; i++) {
    if (i == 0) sort points according to x + y;
    else sort points according to x − y;
    A[1] = A[2] = ∅;
    for each point p in the order {
        find points in A[1], A[2] such that p is in their
            R_{2i+1} and R_{2i+2} regions, respectively;
        connect p with the nearest point in each subset;
        delete the subsets from A[1], A[2], respectively;
        add p to A[1], A[2];
    }
}
```

Rectilinear Spanning Tree, Figure 2
The rectilinear spanning graph algorithm

Theorem 3 *Given n points in the plane, the rectilinear spanning graph algorithm constructs a spanning graph in* $O(n \log n)$ *time, and the number of edges in the graph is* $O(n)$.

Proof The algorithm can be considered as deleting edges from the complete graph. As described, all deleted edges are redundant based on the cycle property. Thus, the output graph of the algorithm will contain at least one rectilinear minimum spanning tree.

In the algorithm, each given point will be inserted and deleted at most once from the active set for each of the four regions R_1 through R_4. For each insertion or deletion, the algorithm requires $O(\log n)$ time. Thus, the total time is upper bounded by $O(n \log n)$. The storage is needed only for active sets, which is at most $O(n)$. □

Applications

Rectilinear minimum spanning tree problem has wide applications in VLSI CAD. It is frequently used as a metric of wire length estimation during placement. It is often constructed to approximate a minimum Steiner tree and is also a key step in many Steiner tree heuristics. It is also used in an approximation to the traveling salesperson problem which can be used to generate scan chains in testing. It is important to emphasize that for real world applications, the input sizes are usually very large. Since it is a problem that will be computed hundreds of thousands times and many of them will have very large input sizes, the rectilinear minimum spanning tree problem needs a very efficient algorithm.

Experimental Results

The experimental results using the Rectilinear Spanning Graph (RSG) followed by Kruskal's algorithm for a rectilinear minimum spanning tree were reported in Zhou

Rectilinear Spanning Tree, Table 1
Experimental Results

Input		Complete		Bound-degree		RSG	
orig	distinct	#edge	time	#edge	time	#edge	time
1000	999	498501	5.095 s	3878	0.299 s	2571	0.112 s
2000	1996	1991010	24.096 s	7825	0.996 s	5158	0.218 s
4000	3995	7978015	2 m 7.233 s	15761	3.452 s	10416	0.337 s
6000	5991	17943045	5 m 54.697 s	23704	7.515 s	15730	0.503 s
8000	7981	31844190	13 m 7.682 s	31624	13.141 s	21149	0.672 s
10000	9962	49615741	–	39510	20.135 s	26332	0.934 s
12000	11948	–	–	47424	32.300 s	31586	1.052 s
14000	13914	–	–	55251	46.842 s	36853	1.322 s
16000	15883	–	–	63089	1 m 3.759 s	42251	1.486 s
18000	17837	–	–	70876	1 m 19.812 s	47511	1.701 s
20000	19805	–	–	78723	1 m 45.792 s	52732	1.907 s

et al. [5]. Two other approaches were compared. The first approach used the complete graph on the point set as the input to Kruskal's algorithm. The second approach is an implementation of concepts described in [3]; namely for each point, scan all other points but only connect the nearest one in each quadrant region. With sizes ranging from 1000 to 20,000, randomly generated point sets were used in the experiments. The results are reproduced here in Table 1. The first column gives the number of generated points; the second column gives the number of distinct points. For each approach, the number of edges in the given graph and the total running time are reported. For input size larger than 10,000, the complete graph approach simply runs out of memory.

Cross References

► Rectilinear Steiner Tree

Recommended Reading

1. McCreight, E.M.: Priority search trees. SIAM J. Comput. **14**, 257–276 (1985)
2. Pugh, W.: Skip lists: A probabilistic alternative to balanced trees. Commun. ACM **33**, 668–676 (1990)
3. Robins, G., Salowe, J.S.: Low-degree minimum spanning tree. Discret. Comput. Geom. **14**, 151–165 (1995)
4. Zheng, S.Q., Lim, J.S., Iyengar, S.S.: Finding obstacle-avoiding shortest paths using implicit connection graphs. IEEE Trans. Comput. Aided Des. **15**, 103–110 (1996)
5. Zhou, H., Shenoy, N., Nicholls, W.: Efficient minimum spanning tree construction without delaunay triangulation. In: Proc. Asian and South Pacific Design Automation Conference, Yokohama, Japan (2001)
6. Zhou, H., Shenoy, N., Nicholls, W.: Efficient spanning tree construction without delaunay triangulation. Inf. Proc. Lett. **81**, 271–276 (2002)

Rectilinear Steiner Tree

2004; Zhou

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

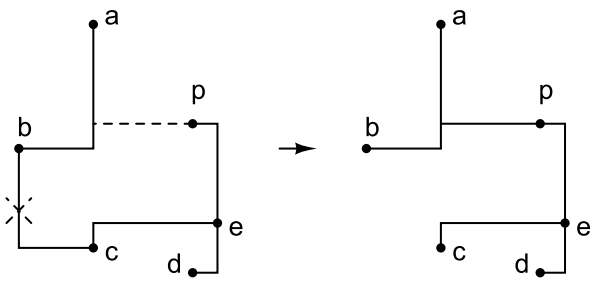
Keywords and Synonyms

Metric minimum Steiner tree; Shortest routing tree

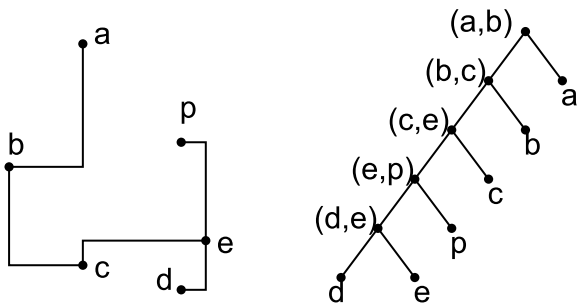
Problem Definition

Given n points on a plane, a Steiner minimal tree connects these points through some extra points (called Steiner points) to achieve a minimal total length. When the length between two points is measured by the rectilinear distance, the tree is called a rectilinear Steiner minimal tree.

Because of its importance, there is much previous work to solve the SMT problem. These algorithms can be grouped into two classes: exact algorithms and heuristic algorithms. Since SMT is NP-hard, any exact algorithm is expected to have an exponential worst-case running time. However, two prominent achievements must be noted in this direction. One is the `GeoSteiner` algorithm and implementation by Warme, Winter, and Zacharisen [14,15], which is the current fastest exact solution to the problem. The other is a Polynomial Time Approximation Scheme (PTAS) by Arora [1], which is mainly of theoretical importance. Since exact algorithms have long running time, especially on large input sizes, much more previous efforts were put on heuristic algorithms. Many of them generate a Steiner tree by improving on a minimal spanning tree topology [7], since it was



Rectilinear Steiner Tree, Figure 1
Edge substitution by Borah et al.



Rectilinear Steiner Tree, Figure 2
A minimal spanning tree and its merging binary tree

proved that a minimal spanning tree is a 3/2 approximation of a SMT [8]. However, since the backbones are restricted to the minimal spanning tree topology in these approaches, there is a reported limit on the improvement ratios over the minimal spanning trees. The iterated 1-Steiner algorithm by Kahng and Robins [10] is an early approach to deviate from that restriction and an improved implementation [6] is a champion among such programs in public domain. However, the implementation in [10] has a running time of $O(n^4 \log n)$ and the implementation in [6] has a running time of $O(n^3)$. A much more efficient approach was later proposed by Borah et al. [2]. In their approach, a spanning tree is iteratively improved by connecting a point to an edge and deleting the longest edge on the created circuit. Their algorithm and implementation had a worst-case running time of $\Theta(n^2)$, even though an alternative $O(n \log n)$ implementation was also proposed. Since the backbone is no longer restricted to the minimal spanning tree topology, its performance was reported to be similar to the iterated 1-Steiner algorithm [2]. A recent effort in this direction is a new heuristic by Mandoiu et al. [11] which is based on a 3/2 approximation algorithm of the metric Steiner tree problem on quasi-bipartite graphs [12]. It performs slightly better than the iterated 1-Steiner algorithm, but its running time is also slightly longer than the iterated 1-Steiner algorithm (with the empty rectangle test [11] used). More recently, Chu [3] and Chu and Wong [4] proposed an efficient lookup table based approach for rectilinear Steiner tree construction.

Key Results

The presented algorithm is based on the edge substitution heuristic of Borah et al. [2]. The heuristic works as follows. It starts with a minimal spanning tree and then iteratively considers connecting a point (for example p in Fig. 1) to a nearby edge (for example (a, b)) and deleting the longest edge ((b, c)) on the circuit thus formed. The algorithm employs the spanning graph [17] as a backbone of the computation: it is first used to generate the initial minimal spanning tree, and then to generate point-edge pairs for tree improvements. This kind of unification happens also in the spanning tree computation and the longest edge computation for each point-edge pair: using Kruskal's algorithm with disjoint set operations (instead of Prim's algorithm) [5] will unify these two computations.

In order to reduce the number of point-edge pair candidates from $O(n^2)$ to $O(n)$, Borah et al. suggested to use the visibility of a point from an edge, that is, only a point visible from an edge can be considered to connect to that edge. This requires a sweepline algorithm to find visibility relations between points and edges. In order to skip this complex step, the geometrical proximity information embedded within the spanning graph is leveraged. Since a point has eight nearest points connected around it, it is observed that if a point is visible to an edge then the point is *usually* connected in the graph to at lease one end point. In the algorithm, the spanning graph is used to generate point-edge pair candidates. For each edge in the current tree, all points that are neighbors of either of the end points will be considered to form point-edge pairs with the edge. Since the cardinality of the spanning graph is $O(n)$, the number of possible point-edge pairs generated in this way is also $O(n)$.

When connecting a point to an edge, the longest edge on the formed circuit needs to be deleted. In order to find the corresponding longest edge for each point-edge pair efficiently, it explores how the spanning tree is formed through Kruskal's algorithm. This algorithm first sorts the edges into non-decreasing lengths and each edge is considered in turn. If the end points of the edge have been connected, then the edge will be excluded from the spanning tree, otherwise, it will be included. The structure of these connecting operations can be represented by a binary tree, where the leaves represent the points and the

```
Rectilinear Steiner Tree (RST) Algorithm
    T = ∅;
    Generate the spanning graph G by RSG algorithm;
    for (each edge (u, v) ∈ G in non-decreasing length) {
        s1 = find_set(u); s2 = find_set(v);
        if (s1 != s2) {
            add (u, v) in tree T;
            for (each neighbor w of u, v in G)
                if (s1 == find_set(w))
                    lca_add_query(w, u, (u, v));
                else lca_add_query(w, v, (u, v));
            lca_tree_edge((u, v), s1.edge);
            lca_tree_edge((u, v), s2.edge);
            s = union_set(s1, s2); s.edge = (u, v);
        }
    }
    generate point-edge pairs by lca_answer_queries;
    for (each pair (p, (a, b), (c, d)) in non-increasing positive gains)
        if ((a, b), (c, d) has not been deleted from T) {
            connect p to (a, b) by adding three edges to T;
            delete (a, b), (c, d) from T;
        }
```

Rectilinear Steiner Tree, Figure 3
The rectilinear Steiner tree algorithm

internal nodes represent the edges. When an edge is included in the spanning tree, a node is created representing the edge and has as its two children the trees representing the two components connected by this edge. To illustrate this, a spanning tree with its representing binary tree are shown in Fig. 2. As can be seen, the longest edge between two points is the least common ancestor of the two points in the binary tree. For example, the longest edge between p and b in Fig. 2 is (b, c), which is the least common ancestor of p and b in the binary tree. To find the longest edge on the circuit formed by connecting a point to an edge, it needs to find the longest edge between the point and one end point of the edge that are in the same component before connecting the edge. For example, consider the pair p and (a, b), since p and b are in the same component before connecting (a, b), the edge needs to be deleted is the longest between p and b.

Based on the above discussion, the pseudo-code of the algorithm can be described in Fig. 3. At the beginning of the algorithm, Zhou et al.'s rectilinear spanning graph algorithm [17] is used to generate the spanning graph G for the given set of points. Then Kruskal's algorithm is used on the graph to generate a minimal spanning tree. The data structure of disjoint sets [5] is used to merge components and check whether two points are in the same component (the first **for** loop). During this process, the merging binary tree and the queries for least common ancestors of all point-edge pairs are also generated. Here s, $s1$, and $s2$ represent disjoint sets and each records the root of the component in the merging binary tree. For each edge (u, v) adding to T, each neighbor w of either u or v will be considered to connect to (u, v). The longest edge for this pair is the least common ancestor of w, u or w, v depending on which point is in the same component as w. The procedure `lca_add_query` is used to add this query. Connecting the two components by (u, v) will also be recorded in the merging binary tree by the procedure `lca_tree_edge`. After generating the minimal spanning tree, it also has the corresponding merging binary tree and the least common ancestor queries ready. Using Tarjan's off-line least common ancestor algorithm [5] (represented by `lca_answer_queries`), it can generate all longest

Rectilinear Steiner Tree, Table 1
Comparison with other algorithms I

Input size	GeoSteiner		BI1S		BOI		RST	
	Improve	Time	Improve	Time	Improve	Time	Improve	Time
100	11.440	0.487	10.907	0.633	9.300	0.0267	10.218	0.004
200	11.492	3.557	10.897	4.810	9.192	0.1287	10.869	0.020
300	11.492	12.685	10.931	18.770	9.253	0.2993	10.255	0.041
500	11.525	72.192	–	–	9.274	0.877	10.381	0.084
800	11.343	536.173	–	–	9.284	2.399	10.719	0.156
1000	–	–	–	–	9.367	4.084	10.433	0.186
2000	–	–	–	–	9.326	31.098	10.523	0.381
3000	–	–	–	–	9.390	104.919	10.449	0.771
5000	–	–	–	–	9.356	307.977	10.499	1.330

Rectilinear Steiner Tree, Table 2
Comparison with other algorithms II

Input size	BGA		Borah		Rohe		RST	
	Improve	Time	Improve	Time	Improve	Time	Improve	Time
Randomly generated testcases								
100	10.272	0.006	10.341	0.004	9.617	0.000	10.218	0.002
500	10.976	0.068	10.778	0.178	10.028	0.010	10.381	0.041
1000	10.979	0.162	10.829	0.689	9.768	0.020	10.433	0.121
5000	11.012	1.695	11.015	25.518	10.139	0.130	10.499	0.980
10000	11.108	4.135	11.101	249.924	10.111	0.310	10.559	2.098
50000	11.120	59.147	–	–	10.109	1.890	10.561	13.029
100000	11.098	161.896	–	–	10.079	4.410	10.514	28.527
500000	–	–	–	–	10.059	27.210	10.527	175.725
VLSI testcases								
337	6.434	0.035	6.503	0.037	5.958	0.010	5.870	0.016
830	3.202	0.070	3.185	0.213	3.102	0.020	2.966	0.033
1944	7.850	0.342	7.772	2.424	6.857	0.040	7.533	0.238
2437	7.965	0.549	7.956	4.502	7.094	0.050	7.595	0.408
2676	8.928	0.623	8.994	3.686	8.067	0.060	8.507	0.463
12052	8.450	4.289	8.465	232.779	7.649	0.300	8.076	2.281
22373	9.848	11.330	9.832	1128.365	8.987	0.570	9.462	4.605
34728	9.046	18.416	9.010	2367.629	8.158	0.900	8.645	5.334

edges for the pairs. With the longest edge for each point-edge pair, the gain of connecting the point to the edge can be calculated. Then each of the point to edge connections will be realized in a non-increasing order of their gains. A connection can only be realized if both the connection edge and deletion edge have not been deleted yet.

The running time of the algorithm is dominated by the spanning graph generation and edge sorting, which take $O(n \log n)$ time. Since the number of edges in the spanning graph is $O(n)$, both Kruskal's algorithm and Tarjan's off-line least common ancestor algorithm take $O(n\alpha(n))$ time, where $\alpha(n)$ is the inverse of Ackermann's function, which grows extremely slow.

Applications

The Steiner Minimal Tree (SMT) problem has wide applications in VLSI CAD. A SMT is generally used in initial topology creation for non-critical nets in physical synthesis. For timing critical nets, minimization of wire length is generally not enough. However, since most nets are non-critical in a design and a SMT gives the most desirable route of such a net, it is often used as an accurate estimation of congestion and wire length during floorplanning and placement. This implies that a Steiner tree algorithm will be invoked millions of times. On the other hand, there exist many large pre-routes in modern VLSI design. The

pre-routes are generally modeled as large sets of points, thus increasing the input sizes of the Steiner tree problem. Since the SMT is a problem that will be computed millions of times and many of them will have very large input sizes, highly efficient solutions with good performance are desired.

Experimental Results

As reported in [16], the first set of experiments were conducted on a Linux system with a 928 MHz Intel Pentium III processor and 512 M memory. The `RST` algorithm was compared with other publicly available programs: the exact algorithm `GeoSteiner` (version 3.1) by Warme, Winter, and Zacharisen [14]; the Batched Iterated 1-Steiner (`BI1S`) by Robins; and the Borah et al.'s algorithm implemented by Madden (`BOI`).

Table 1 gives the results of the first set of experiments. For each input size ranging from 100 to 5000, 30 different test cases are randomly generated through the `rand_points` program in `GeoSteiner`. The improvement ratios of a Steiner tree `St` over its corresponding minimal spanning tree `MST` is defined as $100 \times (\texttt{MST} - \texttt{St})/\texttt{MST}$. For each input size, the average of the improvement ratios and the average running time (in seconds) on each of the programs is reported. As can be seen, `RST` always gives better improvements than `BOI` with less running times.

The second set of experiments compared `RST` with Borah's implementation of Borah et al.'s algorithm (`Borah`), Rohe's Prim-based algorithm (`Rohe`) [13], and Kahng et al.'s Batched Greedy Algorithm (`BGA`) [9]. They were run on a different Linux system with a 2.4 GHz Intel Xeon processor and 2 G memory. Besides the randomly generated test cases, the VLSI industry test cases used in [9] were also used. The results are reported in Table 2.

Cross References

► Rectilinear Spanning Tree

Recommended Reading

1. Arora, S.: Polynomial-time approximation schemes for euclidean tsp and other geometric problem. J. ACM **45**, 753–782 (1998)
2. Borah, M., Owens, R.M., Irwin, M.J.: An edge-based heuristic for steiner routing. IEEE Transac. Comput. Aided Des. **13**, 1563–1568 (1994)
3. Chu, C.: FLUTE: Fast lookup table based wirelength estimation technique. In: Proc. Intl. Conf. on Computer-Aided Design, San Jose, Nov. 2004, pp. 696–701
4. Chu, C., Wong, Y.C.: Fast and accurate rectilinear steiner minimal tree algorithm for vlsi design. In: International Symposium on Physical Design, pp. 28–35 (2005)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge (1989)
6. Griffith, J., Robins, G., Salowe, J.S., Zhang, T.: Closing the gap: Near-optimal steiner trees in polynomial time. IEEE Transac. Comput. Aided Des. **13**, 1351–1365 (1994)
7. Ho, J.M., Vijayan, G., Wong, C.K.: New algorithms for the rectilinear steiner tree problem. IEEE Transac. Comput. Aided Des. **9**, 185–193 (1990)
8. Hwang, F.K.: On Steiner minimal trees with rectilinear distance. SIAM J. Appl. Math. **30**, 104–114 (1976)
9. Kahng, A.B., Mandoiu, I.I., Zelikovsky, A.: Highly scalable algorithms for rectilinear and octilinear steiner trees. In: Proc. Asia and South Pacific Design Automation Conference, Kitakyushu, Japan, (2003) pp. 827–833
10. Kahng, A.B., Robins, G.: A new class of iterative steiner tree heuristics with good performance. IEEE Transac. Comput. Aided Des. **11**, 893–902 (1992)
11. Mandoiu, I.I., Vazirani, V.V., Ganley, J.L.: A new heuristic for rectilinear Steiner trees. In: Proc. Intl. Conf. on Computer-Aided Design, San Jose, (1999)
12. Rajagopalan, S., Vazirani, V.V.: On the bidirected cut relaxation for the metric Steiner tree problem. In: 10th ACM-SIAM Symposium on Discrete Algorithms, Baltimore, (1999), pp. 742–751
13. Rohe, A.: Sequential and Parallel Algorithms for Local Routing. Ph. D. thesis, Bonn University, Bonn, Germany, Dec. (2001)
14. Warme, D.M., Winter, P., Zacharisen, M.: GeoSteiner 3.1 package. ftp://ftp.diku.dk/diku/users/martinz/geosteiner-3.1.tar.gz. Accessed Oct. 2003
15. Warme, D.M., Winter, P., Zacharisen, M.: Exact algorithms for plane steiner tree problems: A computational study, Tech. Rep. DIKU-TR-98/11, Dept. of Computer Science, University of Copenhagen (1998)
16. Zhou, H.: A new efficient retiming algorithm derived by formal manipulation. In: Workshop Notes of Intl. Workshop on Logic Synthesis, Temecula, CA, June (2004)
17. Zhou, H., Shenoy, N., Nicholls, W.: Efficient spanning tree construction without delaunay triangulation. Inf. Process. Lett. **81**, 271–276 (2002)

Registers

1986; Lamport, Vitanyi, Awerbuch

Paul Vitányi
CWI, Amsterdam, Netherlands

Keywords and Synonyms

Shared-memory (wait-free); Wait-free registers; Wait-free shared variables; Asynchronous communication hardware

Problem Definition

Consider a system of asynchronous processes that communicate among themselves by only executing read and write operations on a set of shared variables (also known as shared *registers*). The system has no global clock or other

synchronization primitives. Every shared variable is associated with a process (called *owner*) which writes it and the other processes may read it. An execution of a write (read) operation on a shared variable will be referred to as a *Write* (*Read*) on that variable. A Write on a shared variable puts a value from a pre-determined finite domain into the variable, and a Read reports a value from the domain. A process that writes (reads) a variable is called a *writer* (*reader*) of the variable.

The goal is to construct shared variables in which the following two properties hold. (1) Operation executions are not necessarily atomic, that is, they are not indivisible but rather consist of atomic sub-operations, and (2) every operation finishes its execution within a bounded number of its own steps, irrespective of the presence of other operation executions and their relative speeds. That is, operation executions are *wait-free*. These two properties give rise to a classification of shared variables, depending on their output characteristics. Lamport [8] distinguishes three categories for 1-writer shared variables, using a precedence relation on operation executions defined as follows: for operation executions A and B, A *precedes* B, denoted $A \longrightarrow B$, if A finishes before B starts; A and B *overlap* if neither A precedes B nor B precedes A. In 1-writer variables, all the Writes are totally ordered by "$\longrightarrow$". The three categories of 1-writer shared variables defined by Lamport are the following.

1. A *safe* variable is one in which a Read not overlapping any Write returns the most recently written value. A Read that overlaps a Write may return any value from the domain of the variable.
2. A *regular* variable is a safe variable in which a Read that overlaps one or more Writes returns either the value of the most recent Write preceding the Read or of one of the overlapping Writes.
3. An *atomic* variable is a regular variable in which the Reads and Writes behave as if they occur in some total order which is an extension of the precedence relation.

A shared variable is *boolean*[1] or *multivalued* depending upon whether it can hold only one out of two or one out of more than two values. A *multiwriter* shared variable is one that can be written and read (concurrently) by many processes. If there is only one writer and more than one reader it is called a *multireader* variable.

Key Results

In a series of papers starting in 1974, for details see [4], Lamport explored various notions of concurrent reading and writing of shared variables culminating in the seminal 1986 paper [8]. It formulates the notion of wait-free implementation of an atomic multivalued shared variable—written by a single writer and read by (another) single reader—from safe 1-writer 1-reader 2-valued shared variables, being mathematical versions of physical *flip-flops*, later optimized in [13]. Lamport did not consider constructions of shared variables with more than one writer or reader.

Predating the Lamport paper, in 1983 Peterson [10] published an ingenious wait-free construction of an atomic 1-writer, n-reader m-valued atomic shared variable from $n + 2$ safe 1-writer n-reader m-valued registers, $2n$ 1-writer 1-reader 2-valued atomic shared variables, and 2 1-writer n-reader 2-valued atomic shared variables. He presented also a proper notion of the wait-freedom property. In his paper, Peterson didn't tell how to construct the n-reader boolean atomic variables from flip-flops, while Lamport mentioned the open problem of doing so, and, incidentally, uses a version of Peterson's construction to bridge the algorithmically demanding step from atomic shared bits to atomic shared multivalues. On the basis of this work, N. Lynch, motivated by concurrency control of multi-user data-bases, posed around 1985 the question of how to construct wait-free multiwriter atomic variables from 1-writer multireader atomic variables. Her student Bloom [1] found in 1985 an elegant 2-writer construction, which, however, has resisted generalization to multiwriter. Vitányi and Awerbuch [14] were the first to define and explore the complicated notion of wait-free constructions of general multiwriter atomic variables, in 1986. They presented a proof method, an unbounded solution from 1-writer 1-reader atomic variables, and a bounded solution from 1-writer n-reader atomic variables. The bounded solution turned out not to be atomic, but only achieved regularity ("Errata" in [14]). The paper introduced important notions and techniques in the area, like (bounded) vector clocks, and identified open problems like the construction of atomic wait-free bounded multireader shared variables from flip-flops, and atomic wait-free bounded multiwriter shared variables from the multireader ones. Peterson who had been working on the multiwriter problem for a decade, together with Burns, tried in 1987 to eliminate the error in the unbounded construction of [14] retaining the idea of vector clocks, but replacing the obsolete-information tracking technique by repeated scanning as in [10]. The result [11] was found to be erroneous in the technical report (R. Schaffer, On the correctness of atomic multiwriter registers, Report MIT/LCS/TM-364, 1988). Neither the re-correction in Schaffer's Technical Report, nor the claimed re-correction by the authors of [11] has appeared in print.

[1] Boolean variables are referred to as *bits*.

Also in 1987 there appeared at least five purported solutions for the implementation of 1-writer n-reader atomic shared variable from 1-writer 1-reader ones: [2,7,12] (for the others see [4]) of which [2] was shown to be incorrect (S. Haldar, K. Vidyasankar, *ACM Oper. Syst. Rev*, 26:1(1992), 87–88) and only [12] appeared in journal version. The paper [9], initially a 1987 Harvard Tech Report, resolved all multiuser constructions in one stroke: it constructs a bounded n-writer n-reader (multiwriter) atomic variable from $O(n^2)$ 1-writer 1-reader safe bits, which is optimal, and $O(n^2)$ bit-accesses per Read/Write operation which is optimal as well. It works by making the unbounded solution of [14] bounded, using a new technique, achieving a robust proof of correctness. "Projections" of the construction give specialized constructions for the implementation of 1-writer n-reader (multireader) atomic variables from $O(n^2)$ 1-writer 1-reader ones using $O(n)$ bit accesses per Read/Write operation, and for the implementation of n-writer n-reader (multiwriter) atomic variables from n 1-writer n-reader (multireader) ones. The first "projection" is optimal, while the last "projection" may not be optimal since it uses $O(n)$ control bits per writer while only a lower bound of $\Omega(\log n)$ was established. Taking up this challenge, the construction in [6] claims to achieve this lower bound.

Timestamp System

In a multiwriter shared variable it is only required that every process keeps track of which process wrote last. There arises the general question whether every process can keep track of the order of the last Writes by all processes. A. Israeli and M. Li were attracted to the area by the work in [14], and, in an important paper [5], they raised and solved the question of the more general and universally useful notion of a bounded timestamp system to track the order of events in a concurrent system. In a timestamp system every process owns an *object*, an abstraction of a set of shared variables. One of the requirements of the system is to determine the temporal order in which the objects are written. For this purpose, each object is given a *label* (also referred to as a *timestamp*) which indicates the latest (relative) time when it has been written by its owner process. The processes assign labels to their respective objects in such a way that the labels reflect the real-time order in which they are written to. These systems must support two operations, namely *labeling* and *scan*. A labeling operation execution (Labeling, in short) assigns a new label to an object, and a scan operation execution (Scan, in short) enables a process to determine the ordering in which all the objects are written, that is, it returns a set of labeled-objects ordered temporally. The concern is with those systems where operations can be executed *concurrently*, in an overlapped fashion. Moreover, operation executions must be *wait-free*, that is, each operation execution will take a bounded number of its own steps (the number of accesses to the shared space), irrespective of the presence of other operation executions and their relative speeds. Israeli and Li [5] constructed a bit-optimal bounded timestamp system for *sequential* operation executions. Their sequential timestamp system was published in the above journal reference, but the preliminary concurrent timestamp system in the conference proceedings, of which a more detailed version has been circulated in manuscript form, has not been published in final form. The first generally accepted solution of the *concurrent* case of the bounded timestamp system was from Dolev and Shavit [3]. Their construction is of the type presented in [5] and uses shared variables of size $O(n)$, where n is the number of processes in the system. Each Labeling requires $O(n)$ steps, and each Scan $O(n^2 \log n)$ steps. (A 'step' accesses an $O(n)$ bit variable.) In [4] the unbounded construction of [14] is corrected and extended to obtain an efficient version of the more general notion of a bounded concurrent timestamp system.

Applications

Wait-free registers are, together with message-passing systems, the primary interprocess communication method in distributed computing theory. They form the basis of all constructions and protocols, as can be seen in the textbooks. Wait-free constructions of concurrent timestamp systems (CTSs, in short) have been shown to be a powerful tool for solving concurrency control problems such as various types of mutual exclusion, multiwriter multireader shared variables [14], and probabilistic consensus, by synthesizing a "wait-free clock" to sequence the actions in a concurrent system. For more details see [4].

Open Problems

There is a great deal of work in the direction of register constructions that use less constituent parts, or simpler parts, or parts that can tolerate more complex failures, than previous constructions referred to above. Only, of course, if the latter constructions were not yet optimal in the parameter concerned. Further directions are work on wait-free higher-typed objects, as mentioned above, hierarchies of such objects, and probabilistic constructions. This literature is too vast and diverse to be surveyed here.

Experimental Results

Register constructions, or related constructions for asynchronous interprocess communication, are used in current hardware and software.

Cross References

- ▶ Asynchronous Consensus Impossibility
- ▶ Atomic Broadcast
- ▶ Causal Order, Logical Clocks, State Machine Replication
- ▶ Concurrent Programming, Mutual Exclusion
- ▶ Linearizability
- ▶ Renaming
- ▶ Self-Stabilization
- ▶ Snapshots in Shared Memory
- ▶ Synchronizers, Spanners
- ▶ Topology Approach in Distributed Computing

Recommended Reading

1. Bloom, B.: Constructing two-writer atomic registers. IEEE Trans. Comput. **37**(12), 1506–1514 (1988)
2. Burns, J.E., Peterson, G.L.: Constructing multi-reader atomic values from non-atomic values. In: Proc. 6th ACM Symp. Principles Distr. Comput., pp. 222–231. Vancouver, 10–12 August 1987
3. Dolev, D., Shavit, N.: Bounded concurrent time-stamp systems are constructible. SIAM J. Comput. **26**(2), 418–455 (1997)
4. Haldar, S., Vitanyi, P.: Bounded concurrent timestamp systems using vector clocks. J. Assoc. Comp. Mach. **49**(1), 101–126 (2002)
5. Israeli, A., Li, M.: Bounded time-stamps. Distribut. Comput. **6**, 205–209 (1993) (Preliminary, more extended, version in: Proc. 28th IEEE Symp. Found. Comput. Sci., pp. 371–382, 1987.)
6. Israeli, A., Shaham, A.: Optimal multi-writer multireader atomic register. In: Proc. 11th ACM Symp. Principles Distr. Comput., pp. 71–82. Vancouver, British Columbia, Canada, 10–12 August 1992
7. Kirousis, L.M., Kranakis, E., Vitányi, P.M.B.: Atomic multireader register. In: Proc. Workshop Distributed Algorithms. Lect Notes Comput Sci, vol 312, pp. 278–296. Springer, Berlin (1987)
8. Lamport, L.: On interprocess communication—Part I: Basic formalism, Part II: Algorithms. Distrib. Comput. **1**(2), 77–101 (1986)
9. Li, M., Tromp, J., Vitányi, P.M.B.: How to share concurrent wait-free variables. J. ACM **43**(4), 723–746 (1996) (Preliminary version: Li, M., Vitányi, P.M.B. A very simple construction for atomic multiwriter register. Tech. Rept. TR-01–87, Computer Science Dept., Harvard University, Nov. 1987)
10. Peterson, G.L.: Concurrent reading while writing. ACM Trans. Program. Lang. Syst. **5**(1), 56–65 (1983)
11. Peterson, G.L., Burns, J.E.: Concurrent reading while writing II: The multiwriter case. In: Proc. 28th IEEE Symp. Found. Comput. Sci., pp. 383–392. Los Angeles, 27–29 October 1987
12. Singh, A.K., Anderson, J.H., Gouda, M.G.: The elusive atomic register. J. ACM **41**(2), 311–339 (1994) (Preliminary version in: Proc. 6th ACM Symp. Principles Distribt. Comput., 1987)
13. Tromp, J.: How to construct an atomic variable. In: Proc. Workshop Distrib. Algorithms. Lecture Notes in Computer Science, vol. 392, pp. 292–302. Springer, Berlin (1989)
14. Vitányi, P.M.B., Awerbuch, B.: Atomic shared register access by asynchronous hardware. In: Proc. 27th IEEE Symp. Found. Comput. Sci. pp. 233–243. Los Angeles, 27–29 October 1987. Errata, Proc. 28th IEEE Symp. Found. Comput. Sci., pp. 487–487. Los Angeles, 27–29 October 1987

Regular Expression Indexing

2002; Chan, Garofalakis, Rastogi

Chee-Yong Chan[1], Minos Garofalakis[2], Rajeev Rastogi[3]

[1] Department of Computer Science, National University of Singapore, Singapore, Singapore

[2] Computer Science Division, University of California – Berkeley, Berkeley, CA, USA

[3] Bell Labs, Lucent Technologies, Murray Hill, NJ, USA

Keywords and Synonyms

Regular expression indexing; Regular expression retrieval

Problem Definition

Regular expressions (REs) provide an expressive and powerful formalism for capturing the structure of messages, events, and documents. Consequently, they have been used extensively in the specification of a number of languages for important application domains, including the XPath pattern language for XML documents [6], and the policy language of the *Border Gateway Protocol* (BGP) for propagating routing information between autonomous systems in the Internet [12]. Many of these applications have to manage large databases of RE specifications and need to provide an effective matching mechanism that, given an input string, quickly identifies all the REs in the database that match it. This RE retrieval problem is therefore important for a variety of software components in the middleware and networking infrastructure of the Internet.

The RE retrieval problem can be stated as follows: Given a large set S of REs over an alphabet Σ, where each RE $r \in S$ defines a regular language $L(r)$, construct a data structure on S that efficiently answers the following query: given an arbitrary input string $w \in \Sigma^*$, find the subset S_w of REs in S whose defined regular languages include the string w. More precisely, $r \in S_w$ iff $w \in L(r)$. Since S is a large, dynamic, disk-resident collection of REs, the data

structure should be dynamic and provide efficient support of updates (insertions and deletions) to S. Note that this problem is the opposite of the more traditional RE search problem where $S \subseteq \Sigma^*$ is a collection of strings and the task is to efficiently find all strings in S that match an input regular expression.

Notations

An RE r over an alphabet Σ represents a subset of strings in Σ^* (denoted by $L(r)$) that can be defined recursively as follows [9]: (1) the constants ϵ and $\emptyset$ are REs, where $L(\epsilon) = \{\epsilon\}$ and $L(\emptyset) = \emptyset$; (2) for any letter $a \in \Sigma$, a is a RE where $L(a) = \{a\}$; (3) if r_1 and r_2 are REs, then their union, denoted by $r_1 + r_2$, is a RE where $L(r_1 + r_2) = L(r_1) \cup L(r_2)$; (4) if r_1 and r_2 are REs, then their concatenation, denoted by $r_1.r_2$, is a RE where $L(r_1.r_2) = \{s_1 s_2 \mid s_1 \in L(r_1), s_2 \in L(r_2)\}$; (5) if r is a RE, then its closure, denoted by r^*, is a RE where $L(r^*) = L(\epsilon) \cup L(r) \cup L(rr) \cup L(rrr) \cup \cdots$; and (6) if r is a RE, then a parenthesized r, denoted by (r), is a RE where $L((r)) = L(r)$. For example, if $\Sigma = \{a, b, c\}$, then $(a + b).(a + b + c)^*.c$ is a RE representing the set of strings that begins with either a "a" or a "b" and ends with a "c". A string $s \in \Sigma^*$ is said to match a RE r if $s \in L(r)$.

The language $L(r)$ defined by an RE r can be recognized by a *finite automaton (FA)* M that decides if an input string w is in $L(r)$ by reading each letter in w sequentially and updating its current state such that the outcome is determined by the final state reached by M after w has been processed [9]. Thus, M is an FA for r if the language accepted by M, denoted by $L(M)$, is equal to $L(r)$. An FA is classified as a *deterministic finite automaton* (DFA) if its current state is always updated to a single state; otherwise, it is a *non-deterministic finite automaton* (NFA) if its currant state could refer to multiple possible states. The trade off between a DFA and an NFA representations for a RE is that the latter is more space-efficient while the former is more time-efficient for recognizing a matching string by checking a single path of state transitions. Let $|L(M)|$ denote the size of $L(M)$ and $|L_n(M)|$ denote the number of length-n strings in $L(M)$. Given a set $\mathcal{M}$ of finite automata, let $L(\mathcal{M})$ denote the language recognized by the automata in $\mathcal{M}$; i. e., $L(\mathcal{M}) = \bigcup_{M_i \in \mathcal{M}} L(M_i)$.

Key Results

The RE retrieval problem was first studied for a restricted class of REs in the context of content-based dissemination of XML documents using XPath-based subscriptions (e. g., [1,3,7]), where each XPath expression is processed in terms of a collection of path expressions. While the XPath language [6] allows rich patterns with tree structure to be specified, the path expressions that it supports lack the full expressive power of REs (e. g., XPath does not permit the RE operators *, + and · to be arbitrarily nested in path expressions), and thus extending these XML-filtering techniques to handle general REs may not be straightforward. Further, all of the XPath-based methods are designed for indexing main-memory resident data. Another possible approach would be to coalesce the automata for all the REs into a single NFA, and then use this structure to determine the collection of matching REs. It is unclear, however, if the performance of such an approach would be superior to a simple sequential scan over the database of REs; furthermore, it is not easy to see how such a scheme could be adapted for disk-resident RE data sets.

The first disk-based data structure that can handle the storage and retrieval of REs in their full generality is the *RE-tree* [4,5]. Similar to the R-tree [8], an RE-tree is a dynamic, height-balanced, hierarchical index structure, where the leaf nodes contain data entries corresponding to the indexed REs, and the internal nodes contain "directory" entries that point to nodes at the next level of the index. Each leaf node entry is of the form (*id, M*), where *id* is the unique identifier of an RE r and M is a finite automaton representing r. Each internal node stores a collection of finite automata; and each node entry is of the form (*M, ptr*), where M is a finite automaton and *ptr* is a pointer to some node N (at the next level) such that the following *containment property* is satisfied: If $\mathcal{M}_N$ is the collection of automata contained in node N, then $L(\mathcal{M}_N) \subseteq L(M)$. The automaton M is referred to as the *bounding automaton* for $\mathcal{M}_N$. The containment property is key to improving the search performance of hierarchical index structures like RE-trees: if a query string w is not contained in $L(M)$, then it follows that $w \notin L(M_i)$ for all $M_i \in \mathcal{M}_N$. As a result, the entire subtree rooted at N can be pruned from the search space. Clearly, the closer $L(M)$ is to $L(\mathcal{M}_N)$, the more effective this search-space pruning will be.

In general, there are an infinite number of bounding automata for $\mathcal{M}_N$ with different degrees of precision from the least precise bounding automaton with $L(M) = \Sigma^*$ to the most precise bounding automaton, referred to as the *minimal bounding automaton*, with $L(M) = L(\mathcal{M}_N)$. Since the storage space for an automaton is dependent on its complexity (in terms of the number of its states and transitions), there is a space-precision tradeoff involved in the choice of a bounding automaton for each internal node entry. Thus, even though minimal bounding automata result in the best pruning due to their tightness, it may not be desirable (or even feasible) to always store minimal bounding automata in RE-trees since their space requirement can be too large (possibly exceeding the size of an index node),

thus resulting in an index structure with a low fan-out. Therefore, to maintain a reasonable fan-out for RE-trees, a space constraint is imposed on the maximum number of states (denoted by α) permitted for each bounding automaton in internal RE-tree nodes. The automata stored in RE-tree nodes are, in general, NFAs with a minimum number of states. Also, for better space utilization, each individual RE-tree node is required to contain at least m entries. Thus, the RE-tree height is $O(\log_m(|S|))$.

RE-trees are conceptually similar to other hierarchical, spatial index structures, like the R-tree [8] that is designed for indexing a collection of multi-dimensional rectangles, where each internal entry is represented by a minimal bounding rectangle (MBR) that contains all the rectangles in the node pointed to by the entry. RE-tree search simply proceeds top-down along (possibly) multiple paths whose bounding automaton accepts the input string; RE-tree updates try to identify a "good" leaf node for insertion and can lead to node splits (or, node merges for deletions) that can propagate all the way up to the root. There is, however, a fundamental difference between the RE-tree and the R-tree in the indexed data types: regular languages typically represent *infinite* sets with no well-defined notion of spatial locality. This difference mandates the development of novel algorithmic solutions for the core RE-tree operations. To optimize for search performance, the core RE-tree operations are designed to keep each bounding automaton M in every internal node to be as "tight" as possible. Thus, if M is the bounding automaton for $\mathcal{M}_N$, then $L(M)$ should be as close to $L(\mathcal{M}_N)$ as possible.

There are three core operations that need to be addressed in the RE-tree context: (P1) selection of an optimal insertion node, (P2) computing an optimal node split, and (P3) computing an optimal bounding automaton. The goal of (P1) is to choose an insertion path for a new RE that leads to "minimal expansion" in the bounding automaton of each internal node of the insertion path. Thus, given the collection of automata $\mathcal{M}(N)$ in an internal index node N and a new automaton M, an optimal $M_i \in \mathcal{M}(N)$ needs to be chosen to insert M such that $|L(M_i) \cap L(M)|$ is maximum. The goal of (P2), which arises when splitting a set of REs during an RE-tree node-split, is to identify a partitioning that results in the minimal amount of "covered area" in terms of the languages of the resulting partitions. More formally, given the collection of automata $\mathcal{M} = \{M_1, M_2, \cdots, M_k\}$ in an overflowed index node, find the optimal partition of $\mathcal{M}$ into two disjoint subsets $\mathcal{M}_1$ and $\mathcal{M}_2$ such that $|\mathcal{M}_1| \geq m$, $|\mathcal{M}_2| \geq m$ and $|L(\mathcal{M}_1)| + |L(\mathcal{M}_2)|$ is minimum. The goal of (P3), which arises during insertions, node-splits, or node-merges, is to identify a bounding automaton for a set of REs that does not cover too much "dead space". Thus, given a collection of automata $\mathcal{M}$, the goal is to find the optimal bounding automaton M such that the number of states of M is no more than α, $L(\mathcal{M}) \subseteq L(M)$ and $|L(M)|$ is minimum.

The objective of the above three operations is to maximize the pruning during search by keeping bounding automata tight. In (P1), the optimal automaton M_i selected (within an internal node) to accommodate a newly inserted automaton M is to maximize $|L(M_i) \cap L(M)|$. The set of automata $\mathcal{M}$ are split into two tight clusters in (P2), while in (P3), the most precise automaton (with no more than α states) is computed to cover the set of automata in $\mathcal{M}$. Note that (P3) is unique to RE-trees, while both (P1) and (P2) have their equivalents in R-trees. The heuristics solutions [2,8] proposed for (P1) and (P2) in R-trees aim to minimize the number of visits to nodes that do not lead to any qualifying data entries. Although the minimal bounding automata in RE-trees (which correspond to regular languages) are very different from the MBRs in R-trees, the intuition behind minimizing the area of MBRs (total area or overlapping area) in R-trees should be effective for RE-trees as well. The counterpart for area in an RE-tree is $|L(M)|$, the size of the regular language for M. However, since a regular language is generally an infinite set, new measures need to be developed for the size of a regular language or for comparing the sizes of two regular languages.

One approach to compare the relative sizes of two regular languages is based on the following definition: for a pair of automata M_i and M_j, $L(M_i)$ is said to be larger than $L(M_j)$ if there exists a positive integer N such that for all $k \geq N$, $\sum_{l=1}^{k} |L_l(M_i)| \geq \sum_{l=1}^{k} |L_l(M_j)|$. Based on the above intuition, three increasingly sophisticated measures are proposed to capture the size of an infinite regular language. The *max-count measure* simply counts the number of strings in the language up to a certain size λ; i. e., $|L(M)| = \sum_{i=1}^{\lambda} |L_i(M)|$. This measure is useful for applications where the maximum length of all the REs to be indexed are known and is not too large so that λ can be set to some value slightly larger than the maximum length of the REs. A second more robust measure that is less sensitive to the λ parameter value is the *rate-of-growth measure* which is based on the intuition that a larger language grows at a faster rate than a smaller language. The size of a language is approximated by computing the rate of change of its size from one "window" of lengths to the next consecutive "window" of lengths: if λ is a length parameter that denote the start of the first window and θ is a window-size parameter, then $|L(M)| = \sum_{\lambda+\theta}^{\lambda+2\theta-1} |L_i(M)| / \sum_{\lambda}^{\lambda+\theta-1} |L_i(M)|$. As in

the max-count measure, the parameters λ and θ should be chosen to be slightly greater than the number of states of M to ensure that strings involving a substantial portion of paths, cycles, and accepting states are counted in each window. However, there are cases where the rate-of-growth measure also fails to capture the "larger than" relationship between regular languages [4]. To address some of the shortcomings of the first two metrics, a third information-theoretic measure is proposed that is based on Rissanen's Minimum description length (MDL) principle [11]. The intuition is that if $L(M_i)$ is larger than $L(M_j)$, then the per-symbol-cost of an MDL-based encoding of a random string in $L(M_i)$ using M_i is very likely to be higher than that of a string in $L(M_j)$ using M_j, where the per-symbol-cost of encoding a string $w \in L(M)$ is the ratio of the cost of an MDL-based encoding of w using M to the length of w. More specifically, if $w = w_1.w_2.\cdots.w_n \in L(M)$ and $s_0, s_1, \ldots, s_n$ is the unique sequence of states visited by w in M, then the MDL-based encoding cost of w using M is given by $\sum_{i=0}^{n-1} \lceil \log_2(n_i) \rceil$, where each n_i denotes the number of transitions out of state s_i, and $\log_2(n_i)$ is the number of bits required to specify the transition out of state s_i. Thus, a reasonable measure for the size of a regular language $L(M)$ is the expected per-symbol-cost of an MDL-based encoding for a random sample of strings in $L(M)$.

To utilize the above metrics for measuring $L(M)$, one common operation needed is the computation of $|L_n(M)|$, the number of length-n strings in $L(M)$. While $|L_n(M)|$ can be efficiently computed when M is a DFA, the problem becomes #P-complete when M is an NFA [10]. Two approaches were proposed to approximate $|L_n(M)|$ when N is an NFA [10]. The first approach is an unbiased estimator for $|L_n(M)|$, which can be efficiently computed but can have a very large standard deviation. The second approach is a more accurate randomized algorithm for approximating $|L_n(M)|$ but it is not very useful in practice due to its high time complexity of $O(n^{\log(n)})$. A more practical approximation algorithm with a time complexity of $O(n^2|M|^2 \min\{|\Sigma|, |M|\})$ was proposed in [4].

The RE-tree operations (P1) and (P2) require frequent computations of $|L(M_i \cap M_j)|$ and $|L(M_i \cup M_j)|$ to be performed for pairs of automata M_i, M_j. These computations can adversely affect RE-tree performance since construction of the intersection and union automaton M can be expensive. Furthermore, since the final automaton M may have many more states than the two initial automata M_i and M_j, the cost of measuring $|L(M)|$ can be high. The performance of these computations can, however, be optimized by using sampling. Specifically, if the counts and samples for each $L(M_i)$ are available, then this information can be utilized to derive approximate counts and samples for $L(M_i \cap M_j)$ and $L(M_i \cup M_j)$ without incurring the overhead of constructing the automata $M_i \cap M_j$ and $M_i \cup M_j$ and counting their sizes. The sampling techniques used are based on the following results for approximating the sizes of and generating uniform samples of unions and intersections of arbitrary sets:

Theorem 1 (Chan, Garofalakis, Rastogi, [4]) *Let r_1 and r_2 be uniform random samples of sets S_1 and S_2, respectively.*

1. *$(|r_1 \cap S_2||S_1|)/|r_1|$ is an unbiased estimator of the size of $S_1 \cap S_2$.*
2. *$r_1 \cap S_2$ is a uniform random sample of $S_1 \cap S_2$ with size $|r_1 \cap S_2|$.*
3. *If the sets S_1 and S_2 are disjoint, then a uniform random sample of $S_1 \cup S_2$ can be computed in $O(|r_1| + |r_2|)$ time. If S_1 and S_2 are not disjoint, then an approximate uniform random sample of $S_1 \cup S_2$ can be computed with the same time complexity.*

Applications

The RE retrieval problem also arises in the context of both XML document classification, which identifies matching DTDs for XML documents, as well as BGP routing, which assigns appropriate priorities to BGP advertisements based on their matching routing-system sequences.

Experimental Results

Experimental results with synthetic data sets [5] clearly demonstrate that the RE-tree index is significantly more effective than performing a sequential search for matching REs, and in a number of cases, outperforms sequential search by up to an order of magnitude.

Recommended Reading

1. Altinel, M., Franklin, M.: Efficient filtering of XML documents for selective dissemination of information. In: *Proceedings of 26th International Conference on Very Large Data Bases*, Cairo, Egypt, pp. 53–64. Morgan Kaufmann, Missouri (2000)
2. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-Tree: An efficient and robust access method for points and rectangles. In: *Proceedings of the ACM International Conference on Management of Data*, Atlantic City, New Jersey, pp. 322–331. ACM Press, New York (1990)
3. Chan, C.-Y., Felber, P., Garofalakis, M., Rastogi, R.: Efficient filtering of XML documents with XPath expressions. In: *Proceedings of the 18th International Conference on Data Engineering*, San Jose, California, pp. 235–244. IEEE Computer Society, New Jersey (2002)
4. Chan, C.-Y., Garofalakis, M., Rastogi, R.: RE-Tree: An efficient index structure for regular expressions. In: *Proceedings of 28th International Conference on Very Large Data Bases*, Hong Kong, China, pp. 251–262. Morgan Kaufmann, Missouri (2002)

5. Chan, C.-Y., Garofalakis, M., Rastogi, R.: RE-Tree: An efficient index structure for regular expressions. *VLDB J.* **12**(2), 102–119 (2003)
6. Clark, J., DeRose, S.: XML Path Language (XPath) Version 1.0. W3C Recommendation, http://www.w3.org./TR/xpath, Accessed Nov 1999
7. Diao, Y., Fischer, P., Franklin, M., To, R.: YFilter: Efficient and scalable filtering of XML documents. In: *Proceedings of the 18th International Conference on Data Engineering*, San Jose, California, pp. 341–342. IEEE Computer Society, New Jersey (2002)
8. Guttman, A.: R-Trees: A dynamic index structure for spatial searching. In: *Proceedings of the ACM International Conference on Management of Data*, Boston, Massachusetts, pp. 47–57. ACM Press, New York (1984)
9. Hopcroft, J., Ullman, J.: *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Massachusetts (1979)
10. Kannan, S., Sweedyk, Z., Mahaney, S.: Counting and random generation of strings in regular languages. In: *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, California, pp. 551–557. ACM Press, New York (1995)
11. Rissanen, J.: Modeling by Shortest Data Description. Automatica **14**, 465–471 (1978)
12. Stewart, J.W.: *BGP4, Inter-Domain Routing in the Internet*. Addison Wesley, Massacuhsetts (1998)

Regular Expression Matching

2004; Navarro, Raffinot

Lucian Ilie
Department of Computer Science, University of Western Ontario, London, ON, Canada

Keywords and Synonyms

Automata-based searching

Problem Definition

Given a *text string* T of length n and a *regular expression* R, the **regular expression matching** problem **(REM)** is to find all text positions at which an occurrence of a string in $L(R)$ ends (see below for definitions).

For an alphabet Σ, a *regular expression* R over Σ consists of elements of $\Sigma \cup \{\varepsilon\}$ (ε denotes the empty string) and operators $\cdot$ (concatenation), $|$ (union), and $*$ (iteration, that is, repeated concatenation); the set of strings $L(R)$ represented by R is defined accordingly; see [5]. It is important to distinguish two measures for the size of a regular expression: the *size*, m, which is the total number of characters from $\Sigma \cup \{\cdot, |, *\}$, and Σ*-size*, m_Σ, which counts only the characters in Σ. As an example, for $R = (\mathtt{A}|\mathtt{T})((\mathtt{C}|\mathtt{CG})*)$, the set $L(R)$ contains all strings that start with an A or a T followed by zero or more strings in the set $\{\mathtt{C}, \mathtt{CG}\}$; the size of R is $m = 8$ and the Σ-size is $m_\Sigma = 5$. Any regular expression can be processed in linear time so that $m = \mathcal{O}(m_\Sigma)$ (with a small constant); the difference becomes important when the two sizes appear as exponents.

Key Results

Finite Automata

The classical solutions for the REM problem involve finite automata which are directed graphs with the edges labeled by symbols from $\Sigma \cup \{\varepsilon\}$; their nodes are called states; see [5] for details. Unrestricted automata are called *nondeterministic finite automata (NFA)*. *Deterministic finite automata (DFA)* have no ε-labels and require that no two outgoing edges of the same state have the same label. Regular expressions and DFAs are equivalent, that is, the sets of strings represented are the same, as shown by Kleene [8]. There are two classical ways of computing an NFA from a regular expression. Thompson's construction [14], builds an NFA with up to $2m$ states and up to $4m$ edges whereas Glushkov–McNaughton–Yamada's automaton [3,9] has the minimum number of states, $m_\Sigma + 1$, and $\mathcal{O}(m_\Sigma^2)$ edges; see Fig. 1. Any NFA can be converted into an equivalent DFA by the *subset construction*: each subset of the set of states of the NFA becomes a state of the DFA. The problem is that the DFA can have exponentially more states than the NFA. For instance, the regular expression $((\mathtt{a}|\mathtt{b})*)\mathtt{a}(\mathtt{a}|\mathtt{b})(\mathtt{a}|\mathtt{b})\ldots(\mathtt{a}|\mathtt{b})$, with k occurrences of the $(\mathtt{a}|\mathtt{b})$ term, has a $(k+2)$-state NFA but requires $\Omega(2^k)$ states in any equivalent DFA.

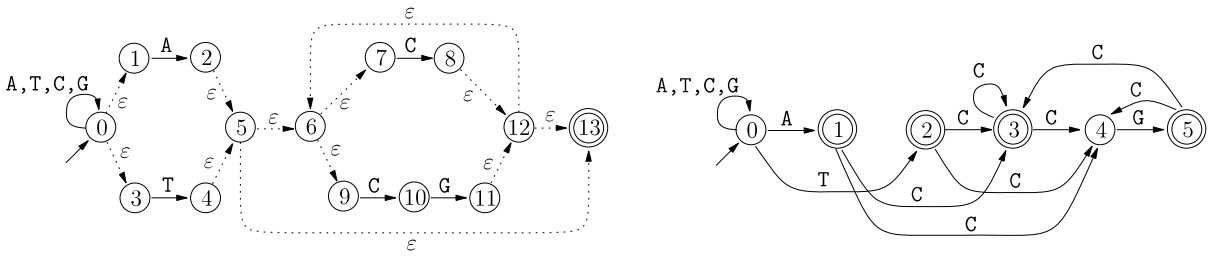
Classical Solutions

A regular expression is first converted into an NFA or DFA which is then simulated on the text. In order to be able to search for a match starting anywhere in the text, a loop labeled by all elements of Σ is added to the initial state; see Fig. 1.

Searching with an NFA requires linear space but many states can be active at the same time and to update them all one needs, for Thompson's NFA, $\mathcal{O}(m)$ time for each letter of the text; this gives Theorem 1. On the other hand, DFAs allow searching time that is linear in n but require more space for the automaton. Theorem 2 uses the DFA obtained from the Glushkov–McNaughton–Yamada's NFA.

Theorem 1 (Thompson [14]) *The REM problem can be solved with an NFA in $\mathcal{O}(mn)$ time and $\mathcal{O}(m)$ space.*

Theorem 2 (Kleene [8]) *The REM problem can be solved with a DFA in $\mathcal{O}(n + 2^{m_\Sigma})$ time and $\mathcal{O}(2^{m_\Sigma})$ space.*



Regular Expression Matching, Figure 1
Thompson's NFA (*left*) and Glushkov–McNaughton–Yamada's NFA (*right*) for the regular expression `(A|T)((C|CG)*)`; the initial loops labeled `A,T,C,G` are not part of the construction, they are needed for REM

Lazy Construction and Modules

One heuristic to alleviate the exponential increase in the size of DFA is to build only the states reached while scanning the text, as implemented in *Gnu Grep*. Still, the space needed for the DFA remains a problem. A four-Russians approach was presented by Myers [10] where a tradeoff between the NFA and DFA approaches is proposed. The syntax tree of the regular expression is divided into modules which are implemented as DFAs and are thereafter treated as leaf nodes in the syntax tree. The process continues until a single module is obtained.

Theorem 3 (Myers [10]) *The REM problem can be solved in* $\mathcal{O}(mn/\log n)$ *time and* $\mathcal{O}(mn/\log n)$ *space.*

Bit-Parallelism

The simulation of the above mentioned modules is done by encoding all states as bits of a single computer word (called *bit mask*) so that all can be updated in a single operation. The method can be used without modules, to simulate directly an NFA as done in [17] and implemented in the *Agrep* software [16]. Note that, in fact, the DFA is also simulated: a whole bit mask corresponds to a subset of states of the NFA, that is, one state of the DFA.

The bit-implementation of Wu and Manber [17] uses the property of Thompson's automaton that all Σ-labeled edges connect consecutive states, that is, they carry a bit 1 from position i to position $i+1$. This makes it easy to deal with the Σ-labeled edges but the ε-labeled ones are more difficult. A table of size linear in the number of states of the DFA needs to be precomputed to account for the ε-closures (set of states reachable from a given state by ε-paths).

Note that in Theorems 1, 2, and 3 the space complexity is given in words. In Theorems 4 and 5 below, for a more practical analysis, the space is given in bits and the alphabet size is also taken into consideration. For comparison, the space in Theorem 2, given in bits, is $\mathcal{O}(|\Sigma| m_\Sigma 2^{m_\Sigma})$.

Theorem 4 (Wu and Manber [17]) *Thompson's automaton can be implemented using* $2m(2^{2m+1} + |\Sigma|)$ *bits.*

Glushkov–McNaughton–Yamada's automaton has different structural properties. First, it is ε-free, that is, there are no ε-labels on edges. Second, all edges incoming to a given state are labeled the same. These properties are exploited by Navarro and Raffinot [13] to construct a bit-parallel implementation that requires less space. The results is a simple algorithm for regular expression searching which uses less space and usually performs faster than any existing algorithm.

Theorem 5 (Navarro and Raffinot [13]) *Glushkov–McNaughton–Yamada's automaton can be implemented using* $(m_\Sigma + 1)(2^{m_\Sigma+1} + |\Sigma|)$ *bits.*

All algorithms in this category run in $\mathcal{O}(n)$ time but smaller DFA representation implies more locality of reference and thus faster algorithms in practice. An improvement of any algorithm using Glushkov–McNaughton–Yamada's automaton can be done by reducing first the automaton by merging some of its states, as done by Ilie et al. [6]. The reduction can be performed in such a way that all useful properties of the automaton are preserved. The search becomes faster due to the reduction in size.

Filtration

The above approaches examine every character in the text. In [15] a multipattern search algorithm is used to search for strings that must appear inside any occurrence of the regular expression. Another technique is used in *Gnu Grep*; it extracts the longest string that must appear in any match (it can be used only when such a string exists). In [13], bit-parallel techniques are combined with a reverse factor search approach to obtain a very fast character skipping algorithm for regular expression searching.

Related Problems

Regular expressions with *backreference* have a feature that helps remembering what was matched to be used later; the matching problem becomes NP-complete; see [1]. *Extended* regular expressions involve adding two extra operators, intersection and complement, which do not change the expressive power. The corresponding matching problem can be solved in $\mathcal{O}((n+m)^4)$ time using dynamic programming, see [5, Exercise 3.23].

Concerning finite automata construction, recall that Thompson's NFA has $\mathcal{O}(m)$ edges whereas the ε-free Glushkov–McNaughton–Yamada's NFA can have a quadratic number of edges. It has been shown in [2] that one can always build an ε-free NFA with $\mathcal{O}(m \log m)$ edges (for fixed alphabets). However, it is the number of states which is more important in the searching algorithms.

Applications

Regular expression matching is a powerful tool in text-based applications, such as text retrieval and text editing, and in computational biology to find various motifs in DNA and protein sequences. See [4] for more details.

Open Problems

The most important theoretical problem is whether linear time and linear space can be achieved simultaneously. Characterizing the regular expressions that can be searched for using a linear-size equivalent DFA is also of interest. The expressions consisting of a single string are included here – the algorithm of Knuth, Morris, and Pratt is based on this. Also, it is not clear how much an NFA can be efficiently reduced (as done by [6]); the problem of finding a minimal NFA is PSPACE-complete, see [7]. Finally, for testing, it is not clear how to define random regular expressions.

Experimental Results

A disadvantage of the bit-parallel technique compared with the classical implementation of a DFA is that the former builds all possible subsets of states whereas the latter builds only the states that can be reached from the initial one (the other ones are useless). On the other hand, bit-parallel algorithms are simpler to code, more flexible (they allow also approximate matching), and there are techniques for reducing the space required. Among the bit-parallel versions, Glushkov–McNaughton–Yamada-based algorithms are better than Thompson-based ones. Modules obtain essentially the same complexity as bit-parallel ones but are more complicated to implement and slower in practice. As the number of computer words increases, bit-parallel algorithms slow down and modules may become attractive. Note also that technological progress has more impact on the bit-parallel algorithms, as opposed to classical ones, since the former depend very much on the machine word size. For details on comparison among various algorithms (including filtration based) see [12]; more recent comparisons are in [13], including the fastest algorithms to date.

URL to Code

Many text editors and programming languages include regular expression search features. They are, as well, among the tools used in protein databases, such as PROSITE and SWISS-PROT, which can be found at http://www.expasy.org/. The package *agrep* [17] can be downloaded from http://webglimpse.net/download.html and *nrgrep* [11] from http://www.dcc.uchile.cl/gnavarro/software.

Cross References

► Approximate Regular Expression Matching is a more general problem where errors are allowed.

Recommended Reading

1. Aho, A.: Algorithms for Finding Patterns in Strings. In: van Leewen, J. (ed.) Handbook of Theoretical Computer Science, vol. A: Algorithms and Complexity, pp. 255–300. Elsevier Science, Amsterdam and MIT Press, Cambridge (1990)
2. Geffert, V.: Translation of binary regular expressions into nondeterministic ε-free automata with $\mathcal{O}(n \log n)$ transitions. J. Comput. Syst. Sci. **66**(3), 451–472 (2003)
3. Glushkov, V.M.: The abstract theory of automata. Russ. Math. Surv. **16**, 1–53 (1961)
4. Gusfield, D.: Algorithms on Strings, Trees and Sequences. Cambridge University Press, Cambridge (1997)
5. Hopcroft, J., Ullman, J.: Introduction to Automata, Languages, and Computation. Addison-Wesley, Reading, MA (1979)
6. Ilie, L., Navarro, G., Yu, S.: On NFA reductions. In: Karhumäki, J. et al. (eds.) Theory is Forever. Lect. Notes Comput. Sci. **3113**, 112–124 (2004)
7. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. SIAM J. Comput. **22**(6), 1117–1141 (1993)
8. Kleene, S.C.: Representation of events in nerve sets. In: Shannon, C.E., McCarthy, J. (eds.) Automata Studies, pp. 3–40. Princeton Univ. Press, Princeton (1956)
9. McNaughton, R., Yamada, H.: Regular expressions and state graphs for automata. IRE Trans. Elect. Comput. **9**(1), 39–47 (1960)
10. Myers, E.: A four Russians algorithm for regular expression pattern matching. J. ACM **39**(2), 430–448 (1992)
11. Navarro, G.: Nr-grep: a fast and flexible pattern matching tool. Softw. Pr. Exp. **31**, 1265–1312 (2001)

12. Navarro, G., Raffinot, M.: Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences. Cambridge University Press, Cambridge (2002)
13. Navarro, G., Raffinot, M.: New techniques for regular expression searching. Algorithmica **41**(2), 89–116 (2004)
14. Thompson, K.: Regular expression search algorithm. Commun. ACM **11**(6), 419–422 (1968)
15. Watson, B.: Taxonomies and Toolkits of Regular Language Algorithms, Ph. D. Dissertation, Eindhoven University of Technology, The Netherlands (1995)
16. Wu, S., Manber, U.: Agrep – a fast approximate pattern-matching tool. In: Proceedings of the USENIX Technical Conf., pp. 153–162 (1992)
17. Wu, S., Manber, U.: Fast text searching allowing errors. Commun. ACM **35**(10), 83–91 (1992)

Reinforcement Learning

1992; Watkins

EYAL EVEN-DAR
Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA

Keywords and Synonyms

Neuro dynamic programming

Problem Definition

Many sequential decision problems ranging from dynamic resource allocation to robotics can be formulated in terms of stochastic control and solved by methods of Reinforcement learning. Therefore, Reinforcement learning (a.k.a Neuro Dynamic Programming) has become one of the major approaches to tackling real life problems.

In Reinforcement learning, an agent wanders in an unknown environment and tries to maximize its long term return by performing actions and receiving rewards. The most popular mathematical models to describe Reinforcement learning problems are the Markov Decision Process (MDP) and its generalization Partially Observable MDP. In contrast to supervised learning, in Reinforcement learning the agent is learning through interaction with the environment and thus influences the "future". One of the challenges that arises in such cases is the exploration-exploitation dilemma. The agent can choose either to exploit its current knowledge and perhaps not learn anything new or to explore and risk missing considerable gains.

While Reinforcement learning contains many problems, due to lack of space this entry focuses on the basic ones. For a detailed history of the development of Reinforcement learning, see [13] chapter 1, the focus of the entry is on Q-learning and Rmax.

Notation

Markov Decision Process: A Markov Decision Process (MDP) formalizes the following problem. An agent is in an environment, which is composed of different states. In each time step the agent performs an action and as a result observes a signal. The signal is composed from the reward to the agent and the state it reaches in the next time step. More formally the MDP is defined as follows,

Definition 1 A Markov Decision process (MDP) M is a 4-tuple (S, A, P, R), where S is a set of the states, A is a set of actions, $P^a_{s,s'}$ is the transition probability from state s to state s' when performing action $a \in A$ in state s, and $R(s, a)$ is the reward distribution when performing action a in state s.

A strategy for an MDP assigns, at each time t, for each state s a probability for performing action $a \in A$, given a history $F_{t-1} = \{s_1, a_1, r_1, \ldots, s_{t-1}, a_{t-1}, r_{t-1}\}$ which includes the states, actions and rewards observed until time $t-1$. While executing a strategy π an agent performs at time t action a_t in state s_t and observe a reward r_t (distributed according to $R(s_t, a_t)$), and a next state s_{t+1} (distributed according to $P^{a_t}_{s_t,\cdot}$). The sequence of rewards is combined into a single value called the *return*. The agent's goal is to maximize the return. There are several natural ways to define the return.

- *Finite horizon:* The return of policy π for a given horizon H is $\sum_{t=0}^{H} r_t$.
- *Discounted return:* For a discount parameter $\gamma \in (0, 1)$, the discounted return of policy π is $\sum_{t=0}^{\infty} \gamma^t r_t$.
- *Undiscounted return:* The return of policy π is $\lim_{t\to\infty} \frac{1}{t+1} \sum_{i=0}^{t} r_i$.

Due to to lack of space, only discounted return, which is the most popular approach mainly due to its mathematical simplicity, is considered. The value function for each state s, under policy π, is defined as $V^\pi(s) = E^\pi[\sum_{i=0}^{\infty} r_i \gamma^i]$, where the expectation is over a run of policy π starting at state s. The state-action value function for using action a in state s and then following π is defined as $Q^\pi(s, a) = R(s, a) + \gamma \sum_{s'} P^a_{s,s'} V^\pi(s')$.

There exists a stationary deterministic optimal policy, π^*, which maximizes the return from any start state [11]. This implies that for any policy π and any state s, $V^{\pi^*}(s) \geq V^\pi(s)$, and $\pi^*(s) = \text{argmax}_a(Q^{\pi^*}(s, a))$. A policy π is ε-optimal if $\| V^{\pi^*} - V^\pi \|_\infty \leq \epsilon$.

Problems Formulation

The Reinforcement learning problems are divided into two categories, planning and learning.

Planning: Given an MDP in its tabular form compute the optimal policy. An MDP is given in its tabular form if the 4-tuple, (A, S, P, R) is given explicitly.

The standard methods for the planning problem in MDP are given below.

Value Iteration: The value iteration is defined as follows. Start with some initial value function, C_s and then iterate using the Bellman operator, $TV(s) = \max_a R(s,a) + \gamma \sum_{s' \in S} P^a_{s,s'} V(s')$.

$$V_0(s) = C_s$$
$$V_{t+1}(s) = TV_t(s)\,,$$

This method relies on the fact that the Bellman operator is contracting. Therefore, the distance between the optimal value function and current value function contracts by a factor of γ with respect to max norm (L_∞) in each iteration.

Policy Iteration: This algorithm starts with initial policy π_0 and iterates over polices. The algorithm has two phases for each iteration. In the first phase, the *Value evaluation step*, a value function for π_t is calculated, by finding the fixed point of $T_{\pi_t} V_{\pi_t} = V_{\pi_t}$, where $T_{\pi_t} V = R(s, \pi_t(s)) + \gamma \sum_{s' \in S} P^{\pi_t(s)}_{s,s'} V(s')$. The second phase, *Policy Improvement step*, is taking the next policy, π_{t+1} as a greedy policy with respect to V_{π_t}. It is known that Policy iteration converges with fewer iterations than value iteration. In practice the convergence of Policy iteration is very fast.

Linear Programming: Formulates and solves an MDP as linear program (LP). The LP variables are $V_1,\ldots,V_n$, where $V_i = V(s_i)$. The definition is:

$$\begin{aligned} &\text{Variables:} && V_1, \ldots, V_n \\ &\text{Minimize:} && \sum_i V_i \\ &\text{Subject to:} && V_i \geq [R(s_i,a) + \gamma \sum_j P_{s_i,s_j}(a) V_j] \\ & && \forall a \in A, s_i \in S. \end{aligned}$$

Learning: Given the states and action identities, learn an (almost)optimal policy through interaction with the environment. The methods are divided into two categories: model free learning and model based learning.

The widely used Q-learning [16] is a model free algorithm. This algorithm belongs to the class of temporal difference algorithms [12]. Q-learning is an off policy method, i. e. it does not depend on the underlying policy and as immediately will be seen it depends on the trajectory and not on the policy generating the trajectory.

Rmax
Set $K = \emptyset$;
if $s \in K$? **then**
  Execute $\hat{\pi}(s)$
else
  Execute a random action;
  if *s becomes known* **then**
    $K = K \bigcup \{s\}$;
    Compute optimal policy, $\hat{\pi}$ for the modified empirical model
  end
end

Reinforcement Learning, Algorithm 1
A model based algorithm

Q learning: The algorithm estimates the state-action value function (for discounted return) as follows:

$$Q_0(s,a) = 0$$
$$Q_{t+1}(s,a) = (1 - \alpha_t(s,a))Q_t(s,a) + \alpha_t(s,a)(r_t(s,a) + \gamma V_t(s'))$$

where s' is the state reached from state s when performing action a at time t, and $V_t(s) = \max_a Q_t(s,a)$. Assume that $\alpha_t(s',a') = 0$ if at time t action a' was not performed at state s'. A learning rate α_t is *well-behaved* if for every state action pair (s,a): (1) $\sum_{t=1}^{\infty} \alpha_t(s,a) = \infty$ and (2) $\sum_{t=1}^{\infty} \alpha_t^2(s,a) < \infty$. As will be seen this is necessary for the convergence of the algorithm.

The model based algorithms are very simple to describe; they simply build an empirical model and use any of the standard methods to find the optimal policy in the empirical (approximate) model. The main challenge in this methods is in balancing exploration and exploitation and having an appropriate stopping condition. Several algorithms give a nice solution for this [3,7]. A version of these algorithms appearing in [6] is described below.

On an intuitive level a state will become known when it was visited "enough" times and one can estimate with high probability its parameters with good accuracy. The modified empirical model is defined as follows. All states that are not in K are represented by a single absorbing state in which the reward is maximal (which causes exploration). The probability to move to the absorbing state from a state $s \in K$ is the empirical probability to move out of K from s and the probability to move between states in K is the empirical probability.

Sample complexity [6] measures how many samples an algorithm need in order to learn. Note that the sample complexity translates into the time needed for the agent to wander in the MDP.

Key Results

The first Theorem shows that the planning problem is easy as long as the MDP is given in its tabular form, and one can use the algorithms presented in the previous section.

Theorem 1 ([10]) *Given an MDP the planning problem is P-complete.*

The learning problem can be done also efficiently using the $R_{\max}$ algorithm as is shown below.

Theorem 2 ([3,7]) *$R_{\max}$ computes an ε-optimal policy from state s with probability at least $1-\delta$ with sample complexity polynomial in $|A|, |S|, \frac{1}{\epsilon}$ and $\log \frac{1}{\delta}$, where s is the state in which the algorithm halts. Also the algorithm's computational complexity is polynomial in $|A|$ and $|S|$.*

The fact that Q-learning converges in the limit to the optimal Q function (which guarantees that the greedy policy with respect to the Q function will be optimal) is now shown.

Theorem 3 ([17]) *If every state-action is visited infinitely often and the learning rate is well behaved then Q_t converges to Q^* with probability one.*

The last statement is regarding the convergence rate of Q-learning. This statement must take into consideration some properties of the underlying policy, and assume that this policy covers the entire state space in reasonable time. The next theorem shows that the convergence rate of Q-learning can vary according to the tuning of the algorithm parameters.

Theorem 4([4]) *Let L be the time needed for the underlying policy to visit every state action with probability 1/2. Let T be the time until $\|Q^* - Q_T\| \leq \epsilon$ with probability at least $1-\delta$ and $\#(s,a,t)$ be the number of times action a was performed at state s until time t. Then if $\alpha_t(s,a) = 1/\#(s,a,t)$, then T is polynomial in $L, \frac{1}{\epsilon}, \log \frac{1}{\delta}$ and exponential in $\frac{1}{1-\gamma}$. If $\alpha_t(s,a) = 1/\#(s,a,t)^{\omega}$ for $\omega \in (1/2, 1)$, then T is polynomial $L, \frac{1}{\epsilon}, \log \frac{1}{\delta}$ and $\frac{1}{1-\gamma}$.*

Applications

The biggest successes of Reinforcement learning so far are mentioned here. For a list of Reinforcement learning successful applications see http://neuromancer.eecs.umich.edu/cgi-bin/twiki/view/Main/SuccessesOfRL.

Backgammon Tesauro [14] used Temporal difference learning combined with neural network to design a player who learned to play backgammon by playing itself, and result in one level with the world's top players.

Helicopter control Ng et al. [9] used inverse Reinforcement learning for autonomous helicopter flight.

Open Problems

While in this entry only MDPs given in their tabular form were discussed much of the research is dedicated to two major directions: large state space and partially observable environments.

In many real world applications, such as robotics, the agent cannot observe the state she is in and can only observes a signal which is correlated with it. In such scenarios the MDP framework is no longer suitable, and another model is in order. The most popular reinforcement learning for such environment is the Partially Observable MDP. Unfortunately, for POMDP even the planning problems are intractable (and not only for the optimal policy which is not stationary but even for the optimal stationary policy); the learning contains even more obstacles as the agent cannot repeat the same state twice with certainty and thus it is not obvious how she can learn. An interesting open problem is trying to characterize when a POMDP is "solvable" and when it is hard to solve according to some structure.

In most applications the assumption that the MDP can be be represented in its tabular form is not realistic and approximate methods are in order. Unfortunately not much theoretically is known under such conditions. Here are a few of the prominent directions to tackle large state space.

Function Approximation: The term function approximation is due to the fact that it takes examples from a desired function (e. g., a value function) and construct an approximation of the entire function. Function approximation is an instance of supervised learning, which is studied in machine learning and other fields. In contrast to the tabular representation, this time a parameter vector Θ represents the value function. The challenge will be to learn the optimal vector parameter in the sense of minimum square error, i. e.

$$\min_{\Theta} \sum_{s \in S} (V^{\pi}(s) - V(s,\Theta))^2,$$

where $V(s,\Theta)$ is the approximation function. One of the most important function approximations is the linear

function approximation,

$$V_t(s, \Theta) = \sum_{i=1}^{T} \phi_s(i)\Theta_t(i) ,$$

where each state has a set of vector features, ϕ_s. A feature based function approximation was analyzed and demonstrated in [2,15]. The main goal here is designing algorithm which converge to almost optimal polices under realistic assumptions.

Factored Markov Decision Process: In a FMDP the set of states is described via a set of random variables $X = \{X_1, \ldots, X_n\}$, where each X_i takes values in some finite domain $Dom(X_i)$. A state s defines a value $x_i \in Dom(X_i)$ for each variable X_i. The transition model is encoded using a dynamic Bayesian network. Although the representation is efficient, not only is finding an ε-optimal policy intractable [8], but it cannot be represented succinctly [1]. However, under few assumptions on the FMDP structure there exists algorithms such as [5] that have both theoretical guarantees and nice empirical results.

Cross References

- ► Attribute-Efficient Learning
- ► Learning Automata
- ► Learning Constant-Depth Circuits
- ► Mobile Agents and Exploration
- ► PAC Learning

Recommended Reading

1. Allender, E., Arora, S., Kearns, M., Moore, C., Russell, A.: Note on the representational incompatabilty of function approximation and factored dynamics. In: Advances in Neural Information Processing Systems 15, 2002
2. Bertsekas, D.P., Tsitsiklis, J. N.: Neuro-Dynamic Programming. Athena Scientific, Belmont (1996)
3. Brafman, R., Tennenholtz, M.: R-max – a general polynomial time algorithm for near optimal reinforcement learning. J. Mach. Learn. Res. **3**, 213–231 (2002)
4. Even-Dar, E., Mansour, Y.: Learning rates for Q-learning. J. Mach. Learn. Res. **5**, 1–25 (2003)
5. Guestrin, C., Koller, D., Parr, R., Venkataraman, S.: Efficient solution algorithms for factored mdps. J. Artif. Intell. Res. **19**, 399–468 (2003)
6. Kakade, S.: On the Sample Complexity of Reinforcement Learning. Ph. D. thesis, University College London (2003)
7. Kearns, M., Singh, S.: Near-optimal reinforcement learning in polynomial time. Mach. Learn. **49**(2–3), 209–232 (2002)
8. Lusena, C., Goldsmith, J., Mundhenk, M.: Nonapproximability results for partially observable markov decision processes. J. Artif. Intell. Res. **14**, 83–103 (2001)
9. Ng, A.Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E.:Inverted autonomous helicopter flight via reinforcement learning. In: International Symposium on Experimental Robotics, 2004
10. Papadimitriu, C.H., Tsitsiklis, J.N.: The complexity of markov decision processes. In: Mathematics of Operations Research, 1987, pp. 441–450.
11. Puterman, M.: Markov Decision Processes. Wiley-Interscience, New York (1994)
12. Sutton, R.: Learning to predict by the methods of temporal differences. Mach. Learn. **3**, 9–44 (1988)
13. Sutton, R., Barto, A.: Reinforcement Learning. An Introduction. MIT Press, Cambridge (1998)
14. Tesauro, G.J.: TD-gammon, a self-teaching backgammon program, achieves a master-level play. Neural Comput. **6**, 215–219 (1996)
15. Tsitsiklis, J.N., Van Roy, B.: Feature-based methods for large scale dynamic programming. Mach. Learn. **22**, 59–94 (1996)
16. Watkins, C.: Learning from Delayed Rewards. Ph. D. thesis, Cambridge University (1989)
17. Watkins, C., Dyan, P.: Q-learning. Mach. Learn. **8**(3/4), 279–292 (1992)

Renaming

1990; Attiya, Bar-Noy, Dolev, Peleg, Reischuk

Maurice Herlihy
Department of Computer Science, Brown University, Providence, RI, USA

Keywords and Synonyms

Wait-free renaming

Problem Definition

Consider a system in which $n + 1$ processes $P_0, \ldots, P_n$ communicate either by message-passing or by reading and writing a shared memory. Processes are *asynchronous*: there is no upper or lower bounds on their speeds, and up to t of them may fail undetectably by halting. In the *renaming task* proposed by Attiya, Bar-Noy, Dolev, Peleg, and Reischuk [1], each process is given a unique *input name* taken from a range $0, \ldots, N$, and chooses a unique *output name* taken from a strictly smaller range $0, \ldots, K$. To rule out trivial solutions, a process's decision function must depend only on input names, not its preassigned identifier (so that P_i cannot simply choose output name i). Attiya et al. showed that the task has no solution when $K = n$, but does have a solution when $K = N + t$. In 1993, Herlihy and Shavit [2] showed that the task has no solution when $K < N + t$.

Vertexes, simplexes, and complexes model decision tasks. (See the companion article entitled ► Topology Approach in Distributed Computing). A process's state at the start or end of a task is represented as a vertex $\vec{v}$ labeled

with that process's identifier, and a value, either input or output: $\vec{v} = \langle P, v_i \rangle$. Two such vertexes are *compatible* if (1) they have distinct process identifiers, and (2) those process can be assigned those values together. For example, in the renaming task, input values are required to be distinct, so two input vertexes are compatible only if they are labeled with distinct process identifiers and distinct input values.

Figure 1 shows the output complex for the three-process renaming task using four names. Notice that the two edges marked A are identical, as are the two edges marked B. By identifying these edges, this task defines a simplicial complex that is topologically equivalent to a torus. Of course, after changing the number of processes or the number of names, this complex is no longer a torus.

Key Results

Theorem 1 *Let S^n be an n-simplex, and S^m a face of S^n. Let S be the complex consisting of all faces of S^m, and $\dot{S}$ the complex consisting of all proper faces of S^m (the boundary complex of S). If $\sigma(\dot{S})$ is a subdivision of $\dot{S}$, and $\phi : \sigma(\dot{S}) \to \mathcal{F}(S)$ a simplicial map, then there exists a subdivision $\tau(S)$ and a simplicial map $\psi : \tau(S) \to \mathcal{F}(S)$ such that $\tau(\dot{S}) = \sigma(\dot{S})$, and ϕ and ψ agree on $\sigma(\dot{S})$.*

Informally, any simplicial map of an m-sphere to $\mathcal{F}$ can be "filled in" to a simplicial map of the $(m+1)$-disk. A *span* for $\mathcal{F}(S^n)$ is a subdivision σ of the input simplex S^n together with a simplicial map $\phi : \sigma(S^n) \to \mathcal{F}(S^n)$ such that for every face S^m of S^n, $\phi : \sigma(S^m) \to \mathcal{F}(S^m)$. Spans are constructed one dimension at a time. For each $\vec{s} = \langle P_i, v_i \rangle \in S^n$, ϕ carries $\vec{s}$ to the solo execution by P_i with input $\vec{v}_i$. For each $S^1 = (\vec{s}_0, \vec{s}_1)$, Theorem 1 implies that $\phi(\vec{s}_0)$ and $\phi(\vec{s}_1)$ can be joined by a path in $\mathcal{F}(S^1)$. For each $S^2 = (\vec{s}_0, \vec{s}_1, \vec{s}_2)$, the inductively constructed spans define each face of the boundary complex $\phi : \sigma(S^1_{ij}) \to \mathcal{F}(S^1)_{ij}$, for $i, j \in \{0, 1, 2\}$. Theorem 1 implies that one can "fill in" this map, extending the subdivision from the boundary complex to the entire complex.

Theorem 2 *If a decision task has a protocol in asynchronous read/write memory, then each input simplex has a span.*

One can restrict attention to protocols that have the property that any process chooses the same name in a solo execution.

Definition 1 A protocol is *comparison-based* if the only operations a process can perform on processor identifiers is to test for equality and order; that is, given two P and Q, a process can test for $P = Q$, $P \leq Q$, and $P \geq Q$, but cannot examine the structure of the identifiers in any more detail.

Lemma 3 *If a wait-free renaming protocol for K names exists, then a comparison-based protocol exists.*

Proof Attiya et al. [1] give a simple comparison-based wait-free renaming protocol that uses $2n+1$ output names. Use this algorithm to assign each process an *intermediate* name, and use that intermediate name as input to the K-name protocol. □

Comparison-based algorithms are *symmetric* on the boundary of the span. Let S^n be an input simplex, $\phi : \sigma(S^n) \to \mathcal{F}(S^n)$ a span, and $\mathcal{R}$ the output complex for $2n$ names. Composing the span map ϕ and the decision map δ yields a map $\sigma(S^n) \to \mathcal{R}$. This map can be simplified by replacing each output name by its parity, replacing the complex $\mathcal{R}$ with the binary n-sphere $\mathcal{B}^n$.

$$\mu : \sigma(S^n) \to \mathcal{B}^n \ . \tag{1}$$

Denote the simplex of $\mathcal{B}^n$ whose values are all zero by 0^n, and all one by 1^n.

Lemma 4 $\mu^{-1}(0^n) = \mu^{-1}(1^n) = \emptyset$.

Proof The range $0, \ldots, 2n-1$ does not contain $n+1$ distinct even names or $n+1$ distinct odd names. □

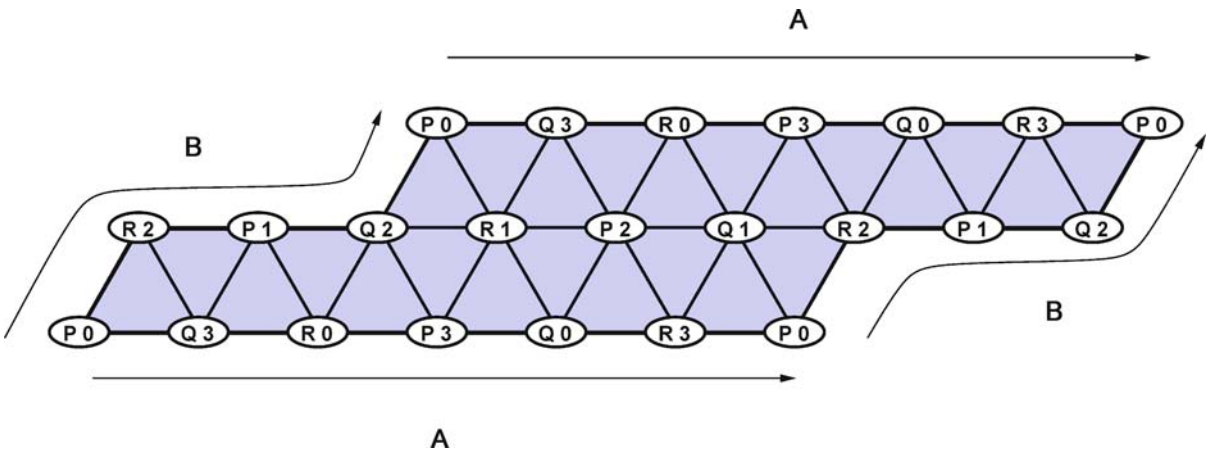
The *n-cylinder* C^n is the binary n-sphere without 0^n and 1^n. Informally, the rest of the argument proceeds by showing that the boundary of the span is "wrapped around" the hole in C^n a non-zero number of times.

The span $\sigma(S^n)$ (indeed any any subdivided n-simplex) is a (combinatorial) *manifold with boundary*: each $(n-1)$-simplex is a face of either one or two n-simplexes. If it is a face of two, then the simplex is an *internal simplex*, and otherwise it is a *boundary* simplex. An orientation of S^n induces an orientation on each n-simplex of $\sigma(S^n)$ so that each internal $(n-1)$-simplex inherits opposite orientations. Summing these oriented simplexes yields a chain, denoted $\sigma_*(S^n)$, such that

$$\partial\sigma_*(S^n) = \sum_{i=0}^{n} (-1)^i \sigma_*(face_i(S^n)) \ .$$

The following is a standard result about the homology of spheres.

Theorem 5 *Let the chain 0^n be the simplex 0^n oriented like S^n. (1) For $0 < m < n$, any two m-cycles are homologous, and (2) every n-cycle C^n is homologous to $k \cdot \partial 0^n$, for some integer k. C^n is a boundary if and only if $k = 0$.*



Renaming, Figure 1
Output complex for 3-process renaming with 4 names

Let S^m be the face of S^n spanned by solo executions of $P_0, \ldots, P_m$. Let 0^m denote some m-simplex of C^n whose values are all zero. Which one will be clear from context.

Lemma 6 *For every proper face S^{m-1} of S^n, there is an m-chain $\alpha(S^{m-1})$ such that*

$$\mu_*(\sigma_*(S^m)) - 0^m - \sum_{i=0}^{m}(-1)^i \alpha(face_i(S^m))$$

is a cycle.

Proof By induction on m. When $m = 1$, $ids(S^1) = \{i, j\}$. 0^1 and $\mu_*(\sigma_*(S^1))$ are 1-chains with a common boundary $\langle P_i, 0\rangle - \langle P_j, 0\rangle$, so $\mu_*(\sigma_*(S^1)) - 0^1$ is a cycle, and $\alpha(\langle P_i, 0\rangle) = \emptyset$.

Assume the claim for $m, 1 \geq m < n - 1$. By Theorem 5, every m-cycle is a boundary (for $m < n - 1$), so there exists an $(m + 1)$-chain $\alpha(S^m)$ such that

$$\mu_*(\sigma_*(S^m)) - 0^m - \sum_{i=0}^{m}(-1)^i \alpha(face_i(S^m)) = \partial\alpha(S^m)\ .$$

Taking the alternating sum over the faces of S^{m+1}, the $\alpha(face_i(S^m))$ cancel out, yielding

$$\mu_*(\partial\sigma_*(S^{m+1})) - \partial 0^{m+1} = \sum_{i=0}^{m+1}(-1)^i \partial\alpha(face_i(S^{m+1}))\ .$$

Rearranging terms yields

$$\partial\left(\mu_*(\sigma_*(S^{m+1})) - 0^{m+1} - \sum_{i=0}^{m+1}(-1)^i \alpha(face_i(S^{m+1}))\right) = 0\ ,$$

implying that

$$\mu_*(\sigma_*(S^{m+1})) - 0^{m+1} - \sum_{i=0}^{m+1}(-1)^i \alpha(face_i(S^{m+1}))$$

is an $(m + 1)$-cycle. □

Theorem 7 *There is no wait-free renaming protocol for $(n + 1)$ processes using $2n$ output names.*

Proof Because

$$\mu_*(\sigma_*(S^{n-1})) - 0^{n-1} - \sum_{i=0}^{n}(-1)^i \alpha(face_i(S^{n-1}))$$

is a cycle, Theorem 5 implies that it is homologous to $k \cdot \partial 0^n$, for some integer k. Because μ is symmetric on the boundary of $\sigma(S^n)$, the alternating sum over the $(n - 1)$-dimensional faces of S^n yields:

$$\mu_*(\partial\sigma_*(S^n)) - \partial 0^n \sim (n + 1)k \cdot \partial 0^n$$

or

$$\mu_*(\partial\sigma_*(S^n)) \sim (1 + (n + 1)k) \cdot \partial 0^n\ .$$

Since there is no value of k for which $(1 + (n + 1)k)$ is zero, the cycle $\mu_*(\partial\sigma_*(S^n))$ is not a boundary, a contradiction. □

Applications

The renaming problem is a key tool for understanding the power of various asynchronous models of computation.

Open Problems

Characterizing the full power of the topological approach to proving lower bounds remains an open problem.

Cross References

- ▶ Asynchronous Consensus Impossibility
- ▶ Set Agreement
- ▶ Topology Approach in Distributed Computing

Recommended Reading

1. Attiya, H., Bar-Noy, A., Dolev, D., Peleg, D., Reischuk, R.: Renaming in an asynchronous environment. J. ACM **37**(3), 524–548 (1990)
2. Herlihy, M.P., Shavit, N.: The asynchronous computability theorem for *t*-resilient tasks. In: Proceedings 25th Annual ACM Symposium on Theory of Computing, 1993, pp. 111–120

Response Time

- ▶ Minimum Flow Time
- ▶ Shortest Elapsed Time First Scheduling

Reversal Distance

- ▶ Sorting Signed Permutations by Reversal (Reversal Distance)

RNA Secondary Structure Boltzmann Distribution

2005; Miklós, Meyer, Nagy

Rune B. Lyngsø
Department of Statistics, Oxford University, Oxford, UK

Keywords and Synonyms

Full partition function

Problem Definition

This problem is concerned with computing features of the Boltzmann distribution over RNA secondary structures in the context of the standard Gibbs free energy model used for RNA Secondary Structure Prediction by Minimum Free Energy (cf. corresponding entry). Thermodynamics state that for a system with configuration space Ω and free energy given by $E\colon \Omega \mapsto \mathbf{R}$, the probability of the system being in state $\omega \in \Omega$ is proportional to $e^{-E(\omega)/RT}$ where R is the universal gas constant and T the absolute temperature of the system. The normalizing factor

$$Z = \sum_{\omega \in \Omega} e^{-E(\omega)/RT} \tag{1}$$

is called the *full partition function* of the system.

Over the past several decades, a model approximating the free energy of a structured RNA molecule by independent contributions of its secondary structure components has been developed and refined. The main purpose of this work has been to assess the stability of individual secondary structures. However, it immediately translates into a distribution over all secondary structures. Early work focused on computing the pairing probability for all pairs of bases, i. e. the sum of the probabilities of all secondary structures containing that base pair. Recent work has extended methods to compute probabilities of base pairing probabilities for RNA heterodimers [2], i. e. interacting RNA molecules, and expectation, variance and higher moments of the Boltzmann distribution.

Notation

Let $s \in \{A, C, G, U\}^*$ denote the sequence of bases of an RNA molecule. Use $X \cdot Y$ where $X, Y \in \{A, C, G, U\}$ to denote a base pair between bases of type X and Y, and $i \cdot j$ where $1 \leq i < j \leq |s|$ to denote a base pair between bases $s[i]$ and $s[j]$.

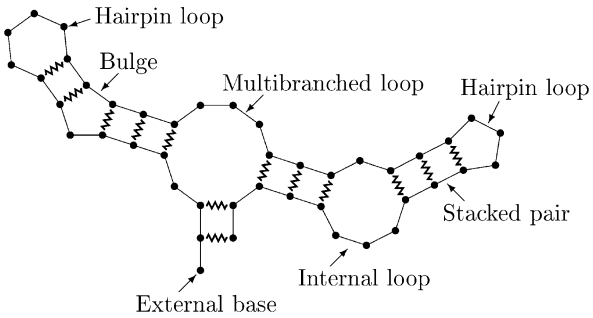
Definition 1 (RNA Secondary Structure) A secondary structure for an RNA sequence s is a set of base pairs $S = \{i \cdot j \mid 1 \leq i < j \leq |s| \wedge i < j - 3\}$. For $i \cdot j, i' \cdot j' \in S$ with $i \cdot j \neq i' \cdot j'$

- $\{i, j\} \cap \{i', j'\} = \emptyset$ (each base pairs with at most one other base)
- $\{s[i], s[j]\} \in \{\{A, U\}, \{C, G\}, \{G, U\}\}$ (only Watson-Crick and G, U wobble base pairs)
- $i < i' < j \Rightarrow j' < j$ (base pairs are either nested or juxtaposed but not overlapping)

The second requirement, that only canonical base pairs are allowed, is standard but not consequential in solutions to the problem. The third requirement states that the structure does not contain pseudoknots. This restriction is crucial for the results listed in this entry.

Energy Model

The model of Gibbs free energy applied, usually referred to as the nearest-neighbor model, was originally proposed by Tinoco et al. [10,11]. It approximates the free energy by postulating that the energy of the full three dimensional



RNA Secondary Structure Boltzmann Distribution, Figure 1
A hypothetical RNA structure illustrating the different loop types. Bases are represented by *circles*, the RNA backbone by *straight lines*, and base pairs by *zigzagged lines*

structure only depends on the secondary structure, and that this in turn can be broken into a sum of independent contributions from each loop in the secondary structure.

Definition 2 (Loops) For $i \cdot j \in S$, base k is *accessible* from $i \cdot j$ iff $i < k < j$ and $\neg\exists i' \cdot j' \in S$: $i < i' < k < j' < j$. The *loop closed by* $i \cdot j$, $\ell_{i \cdot j}$, consists of $i \cdot j$ and all the bases accessible from $i \cdot j$. If $i' \cdot j' \in S$ and i' and j' are accessible from $i \cdot j$, then $i' \cdot j'$ is an interior base pair in the loop closed by $i \cdot j$.

Loops are classified by the number of interior base pairs they contain:

- hairpin loops have no interior base pairs
- stacked pairs, bulges, and internal loops have one interior base pair that is separated from the closing base pair on neither side, on one side, or on both sides, respectively
- multibranched loops have two or more interior base pairs

Bases not accessible from any base pair are called external. This is illustrated in Fig. 1. The free energy of structure S is

$$\Delta G(S) = \sum_{i \cdot j \in S} \Delta G(\ell_{i \cdot j}) \tag{2}$$

where $\Delta G(\ell_{i \cdot j})$ is the free energy contribution from the loop closed by $i \cdot j$. The contribution of S to the full partition function is

$$e^{-\Delta G(S)/RT} = e^{-\sum_{i \cdot j \in S} \Delta G(\ell_{i \cdot j})/RT} = \prod_{\ell_{i \cdot j} \in S} e^{-\Delta G(\ell_{i \cdot j})/RT} . \tag{3}$$

Problem 1 (RNA Secondary Structure Distribution)
INPUT: *RNA sequence s, absolute temperature T and specification of ΔG at T for all loops.*
OUTPUT: $\sum_S e^{-\Delta G(S)/RT}$, *where the sum is over all secondary structures for s.*

Key Results

Solutions are based on recursions similar to those for RNA Secondary Structure Prediction by Minimum Free Energy, replacing sum and minimization with multiplication and sum (or more generally with a *merge function* and a *choice function* [8]). The key difference is that recursions are required to be non-redundant, i. e. any particular secondary structure only contributes through one path through the recursions.

Theorem 1 *Using the standard thermodynamic model for RNA secondary structures, the partition function can be computed in time $O(|s|^3)$ and space $O(|s|^2)$. Moreover, the computation can build data structures that allow $O(1)$ queries of the pairing probability of $i \cdot j$ for any $1 \le i < j \le |s|$ [5,6,7].*

Theorem 2 *Using the standard thermodynamic model for RNA secondary structures, the expectation and variance of free energy over the Boltzmann distribution can be computed in time $O(|s|^3)$ and space $O(|s|^2)$. More generally, the kth moment*

$$E_{\text{Boltzmann}}[\Delta G] = 1/Z \sum_S e^{-\Delta G(S)/RT} \Delta G^k(S) , \tag{4}$$

where $Z = \sum_S e^{-\Delta G(S)/RT}$ is the full partition function and the sums are over all secondary structures for s, can be computed in time $O(k^2|s|^3)$ and space $O(k|s|^2)$ [8].

In Theorem 2 the free energy does not hold a special place. The theorem holds for any function Φ defined by an independent contribution from each loop,

$$\Phi(S) = \sum_{i \cdot j \in S} \phi\left(\ell_{i \cdot j}\right) , \tag{5}$$

provided each loop contribution can be handled with the same efficiency as the free energy contributions. Hence, moments over the Boltzmann distribution of e. g. number of base pairs, unpaired bases, or loops can also be efficiently computed by applying appropriately chosen indicator functions.

Applications

The original use of partition function computations was for discriminating between well defined and less well defined regions of a secondary structure. Minimum free energy predictions will always return a structure. Base pairing probabilities help identify regions where the prediction is uncertain, either due to the approximations of the

model or that the real structure indeed does fluctuate between several low energy alternatives. Moments of Boltzmann distributions are used in identifying how biological RNA molecules deviates from random RNA sequences.

The data structures computed in Theorem 1 can also be used to efficiently sample secondary structures from the Boltzmann distribution. This has been used for probabilistic methods for secondary structure prediction, where the centroid of the most likely cluster of sampled structures is returned rather than the most likely, i. e. minimum free energy, structure [3]. This approach better accounts for the entropic effects of large neighborhoods of structurally and energetically very similar structures. As a simple illustration of this effect, consider twice flipping a coin with probability $p > 0.5$ for heads. The probability p^2 of heads in both flips is larger than the probability $p(1-p)$ of heads followed by tails or tails followed by heads (which again is larger than the probability $(1-p)^2$ of tails in both flips). However, if the order of the flips is ignored the probability of one heads and one tails is $2p(1-p)$. The probability of two heads remains p^2 which is smaller than $2p(1-p)$ when $p < \frac{2}{3}$. Similarly a large set of structures with fairly low free energy may be more likely, when viewed as a set, than a small set of structures with very low free energy.

Open Problems

As for RNA Secondary Structure Prediction by Minimum Free Energy, improvements in time and space complexity are always relevant. This may be more difficult for computing distributions, as the more efficient dynamic programming techniques of [9] cannot be applied. In the context of genome scans, the fact that the start and end positions of encoded RNA molecule is unknown has recently been considered [1].

Also the problem of including structures with pseudoknots, i. e. structures violating the last requirement in Def. 1, in the configuration space is an active area of research. It can be expected that all the methods of Theorems 3 through 6 in the entry on RNA Secondary Structure Prediction Including Pseudoknots can be modified to computation of distributions without affecting complexities. This may require some further bookkeeping to ensure non-redundancy of recursions, and only in [4] has this actively been considered.

Though the moments of functions that are defined as sums over independent loop contributions can be computed efficiently, it is unknown whether the same holds for functions with more complex definitions. One such function that has traditionally been used for statistics on RNA secondary structure [12] is the *order* of a secondary structure which refers to the nesting depth of multibranched loops.

URL to Code

Software for partition function computation and a range of related problems is available from www.bioinfo.rpi.edu/applications/hybrid/download.php and www.tbi.univie.ac.at/~ivo/RNA/. Software including a restricted class of structures with pseudoknots [4] is available at www.nupack.org.

Cross References

- ▶ RNA Secondary Structure Prediction Including Pseudoknots
- ▶ RNA Secondary Structure Prediction by Minimum Free Energy

Recommended Reading

1. Bernhart, S., Hofacker, I.L., Stadler, P.: Local RNA base pairing probabilities in large sequences. Bioinformatics **22**, 614–615 (2006)
2. Bernhart, S.H., Tafer, H., Mückstein, U., Flamm, C., Stadler, P.F., Hofacker, I.L.: Partition function and base pairing probabilities of RNA heterodimers. Algorithms Mol. Biol. **1**, 3 (2006)
3. Ding, Y., Chan, C.Y., Lawrence, C.E.: RNA secondary structure prediction by centroids in a Boltzmann weighted ensemble. RNA **11**, 1157–1166 (2005)
4. Dirks, R.M., Pierce, N.A.: A partition function algorithm for nucleic acid secondary structure including pseudoknots. J. Comput. Chem. **24**, 1664–1677 (2003)
5. Hofacker, I.L., Stadler, P.F.: Memory efficient folding algorithms for circular RNA secondary structures. Bioinformatics **22**, 1172–1176 (2006)
6. Lyngsø, R.B., Zuker, M., Pedersen, C.N.S.: Fast evaluation of internal loops in RNA secondary structure prediction. Bioinformatics **15**, 440–445 (1999)
7. McCaskill, J.S.: The equilibrium partition function and base pair binding probabilities for RNA secondary structure. Biopolymers **29**, 1105–1119 (1990)
8. Miklós, I., Meyer, I.M., Nagy, B.: Moments of the Boltzmann distribution for RNA secondary structures. Bull. Math. Biol. **67**, 1031–1047 (2005)
9. Ogurtsov, A.Y., Shabalina, S.A., Kondrashov, A.S., Roytberg, M.A.: Analysis of internal loops within the RNA secondary structure in almost quadratic time. Bioinformatics **22**, 1317–1324 (2006)
10. Tinoco, I., Borer, P.N., Dengler, B., Levine, M.D., Uhlenbeck, O.C., Crothers, D.M., Gralla, J.: Improved estimation of secondary structure in ribonucleic acids. Nature New Biol. **246**, 40–41 (1973)
11. Tinoco, I., Uhlenbeck, O.C., Levine, M.D.: Estimation of secondary structure in ribonucleic acids. Nature **230**, 362–367 (1971)
12. Waterman, M.S.: Secondary structure of single-stranded nucleic acids. Adv. Math. Suppl. Stud. **1**, 167–212 (1978)

RNA Secondary Structure Prediction Including Pseudoknots

2004; Lyngsø

Rune B. Lyngsø
Department of Statistics, Oxford University, Oxford, UK

Keywords and Synonyms

Abbreviated as *Pseudoknot Prediction*

Problem Definition

This problem is concerned with predicting the set of base pairs formed in the native structure of an RNA molecule, including overlapping base pairs also known as pseudoknots. Standard approaches to RNA secondary structure prediction only allow sets of base pairs that are hierarchically nested. Though few known real structures require the removal of more than a small percentage of their base pairs to meet this criteria, a significant percentage of known real structures contain at least a few base pairs overlapping other base pairs. Pseudoknot substructures are known to be crucial for biological function in several contexts. One of the more complex known pseudoknot structures is illustrated in Fig. 1

Notation

Let $s \in \{A, C, G, U\}^*$ denote the sequence of bases of an RNA molecule. Use $X \cdot Y$ where $X, Y \in \{A, C, G, U\}$ to denote a base pair between bases of type X and Y, and $i \cdot j$ where $1 \leq i < j \leq |s|$ to denote a base pair between bases $s[i]$ and $s[j]$.

Definition 1 (RNA Secondary Structure) A secondary structure for an RNA sequence s is a set of base pairs $S = \{i \cdot j \mid 1 \leq i < j \leq |s| \wedge i < j - 3\}$. For $i \cdot j, i' \cdot j' \in S$ with $i \cdot j \neq i' \cdot j'$

- $\{i, j\} \cap \{i', j'\} = \emptyset$ (each base pairs with at most one other base)
- $\{s[i], s[j]\} \in \{\{A, U\}, \{C, G\}, \{G, U\}\}$ (only Watson-Crick and G, U wobble base pairs)

The second requirement, that only canonical base pairs are allowed, is standard but not consequential in solutions to the problem.

Scoring Schemes

Structures are usually assessed by extending the model of Gibbs free energy used for RNA Secondary Structure Prediction by Minimum Free Energy (cf. corresponding entry) with *ad hoc* extrapolation of multibranched loop energies to pseudoknot substructures [11], or by summing independent contributions e. g. obtained from base pair restricted minimum free energy structures from each base pair [13]. To investigate the complexity of pseudoknot prediction the following three simple scoring schemes will also be considered:

Number of Base Pairs,

$$\#BP(S) = |S|$$

Number of Stacking Base Pairs

$$\#SBP(S) = |\{i \cdot j \in S \mid i + 1 \cdot j - 1 \in S \vee i - 1 \cdot j + 1 \in S\}|$$

Number of Base Pair Stackings

$$\#BPS(S) = |\{i \cdot j \in S \mid i + 1 \cdot j - 1 \in S\}|$$

These scoring schemes are inspired by the fact that stacked pairs are essentially the only loops having a stabilizing contribution in the Gibbs free energy model.

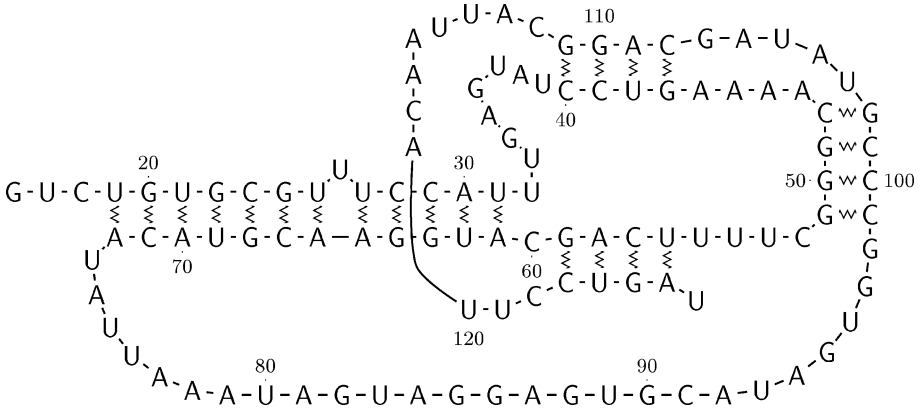
Problem 1 (Pseudoknot Prediction)
Input: *RNA sequence s and an appropriately specified scoring scheme.*
Output: *A secondary structure S for s that is optimal under the scoring scheme specified.*

Key Results

Theorem 1 *The complexities of pseudoknot prediction under the three simplified scoring schemes can be classified as follows, where Σ denotes the alphabet.*

	Fixed alphabet	*Unbounded alphabet*
#BP [13]	*Time* $O\left(|s|^3\right)$, *space* $O\left(|s|^2\right)$	*Time* $O\left(|s|^3\right)$, *space* $O\left(|s|^2\right)$
#SBP [7]	*Time* $O\left(|s|^{1+|\Sigma|^2+|\Sigma|^3}\right)$, *space* $O\left(|s|^{|\Sigma|^2+|\Sigma|^3}\right)$	**NP** *hard*
#BPS	**NP** *hard for* $|\Sigma| = 2$, *PTAS [7]* 1/3-*approximation in time* $O\left(|s|\right)$ *[6]*	**NP** *hard [7],* 1/3-*approximation in time and space* $O\left(|s|^2\right)$ *[6]*

Theorem 2 *If structures are restricted to be planar, i. e. the graph with the bases of the sequence as nodes and base pairs and backbone links of consecutive bases as edges is required to be planar, pseudoknot prediction under the #BPS scoring scheme is* **NP** *hard for an alphabet of size 4. Conversely, a* 1/2-*approximation can be found in time* $O\left(|s|^3\right)$



RNA Secondary Structure Prediction Including Pseudoknots, Figure 1
Secondary structure of the *Escherichia coli* α operon mRNA from position 16 to position 127, cf. [5], Figure 1. The backbone of the RNA molecule is drawn as *straight lines* while base pairings are shown with *zigzagged lines*

and space $O\left(|s|^2\right)$ by observing that an optimal pseudoknot free structure is a 1/2-approximation [6].

There are no steric reasons that RNA secondary structures should be planar, and the structure in Fig. 1 is actually non-planar. Nevertheless, known real structures have relatively simple overlapping base pair patterns with very few non-planar structures known. Hence, planarity has been used as a defining restriction on pseudoknotted structures [2,15]. Similar reasoning has lead to development of several algorithms for finding an optimal structure from restricted classes of structures. These algorithms tend to use more realistic scoring schemes, e. g. extensions of the Gibbs free energy model, than the three simple scoring schemes considered above.

Theorem 3 *Pseudoknot prediction for a restricted class of structures including Fig. 2a through Fig. 2e, but not Fig. 2f, can be done in time $O\left(|s|^6\right)$ and space $O\left(|s|^4\right)$* [11].

Theorem 4 *Pseudoknot prediction for a restricted class of planar structures including Fig. 2a through Fig. 2c, but not Fig. 2d through Fig. 2f, can be done in time $O\left(|s|^5\right)$ and space $O\left(|s|^4\right)$* [14].

Theorem 5 *Pseudoknot prediction for a restricted class of planar structures including Fig. 2a and Fig. 2b, but not Fig. 2c through Fig. 2f, can be done in time $O\left(|s|^5\right)$ and space $O\left(|s|^4\right)$ or $O\left(|s|^3\right)$* [1,4] *(methods differ in generality of scoring schemes that can be used).*

Theorem 6 *Pseudoknot prediction for a restricted class of planar structures including Fig. 2a, but not Fig. 2b through Fig. 2f, can be done in time $O\left(|s|^4\right)$ and space $O\left(|s|^2\right)$* [1,8].

Theorem 7 *Recognition of structures belonging to the restricted classes of Theorems 3, 5, and 6, and enumeration of all irreducible cycles (i. e. loops) in such structures can be done in time $O\left(|s|\right)$* [3,9].

Applications

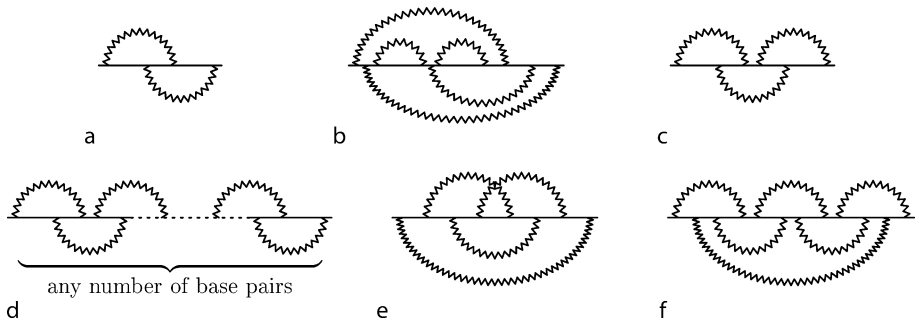
As for the prediction of RNA secondary structures without pseudoknots, the key application of these algorithms are for predicting the secondary structure of individual RNA molecules. Due to the steep complexities of the algorithms of Theorems 3 through 6, these are less well suited for genome scans than prediction without pseudoknots.

Enumerating all loops of a structure in linear time also allows scoring a structure in linear time, as long as the scoring scheme allows the score of a loop to be computed in time proportional to its size. This has practical applications in heuristic searches for good structures containing pseudoknots.

Open Problems

Efficient algorithms for prediction based on restricted classes of structures with pseudoknots that still contain a significant fraction of all known structures is an active area of research. Even using the more theoretical simple *#SBP* scoring scheme, developing e. g. an $O\left(|s|^{|\Sigma|}\right)$ algorithm for this problem would be of practical significance. From a theoretical point of view, the complexity of planar structures is the least well understood, with results for only the *#BPS* scoring scheme.

Classification of and realistic energy models for RNA secondary structures with pseudoknots are much less developed than for RNA secondary structures without pseudoknots. Several recent papers have been addressing this gap [3,9,12].



RNA Secondary Structure Prediction Including Pseudoknots, Figure 2
RNA secondary structures illustrating restrictions of pseudoknot prediction algorithms. Backbone is drawn as a *straight line* while base pairings are shown with *zigzagged arcs*

Data Sets

PseudoBase at http://biology.leidenuniv.nl/~batenburg/PKB.html is a repository of representatives of most known RNA structures with pseudoknots.

URL to Code

The method of Theorem 3 is available at http://selab.janelia.org/software.html#pknots, of one of the methods of Theorem 5 at http://www.nupack.org, and an implementation applying a slight heuristic reduction of the class of structures considered by the method of Theorem 6 is available at http://bibiserv.techfak.uni-bielefeld.de/pknotsrg/ [10].

Cross References

► RNA Secondary Structure Prediction by Minimum Free Energy

Recommended Reading

1. Akutsu, T.: Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. Discret. Appl. Math. **104**, 45–62 (2000)
2. Brown, M., Wilson, C.: RNA pseudoknot modeling using intersections of stochastic context free grammars with applications to database search. In: Hunter, L., Klein, T. (eds.) Proceedings of the 1st Pacific Symposium on Biocomputing, 1996, pp. 109–125
3. Condon, A., Davy, B., Rastegari, B., Tarrant, F., Zhao, S.: Classifying RNA pseudoknotted structures. Theor. Comput. Sci. **320**, 35–50 (2004)
4. Dirks, R.M., Pierce, N.A.: A partition function algorithm for nucleic acid secondary structure including pseudoknots. J. Comput. Chem. **24**, 1664–1677 (2003)
5. Gluick, T.C., Draper, D.E.: Thermodynamics of folding a pseudoknotted mRNA fragment. J. Mol. Biol. **241**, 246–262 (1994)
6. Ieong, S., Kao, M.-Y., Lam, T.-W., Sung, W.-K., Yiu, S.-M.: Predicting RNA secondary structures with arbitrary pseudoknots by maximizing the number of stacking pairs. In: Proceedings of the 2nd Symposium on Bioinformatics and Bioengineering, 2001, pp. 183–190
7. Lyngsø, R.B.: Complexity of pseudoknot prediction in simple models. In: Proceedings of the 31th International Colloquium on Automata, Languages and Programming (ICALP), 2004, pp. 919–931
8. Lyngsø, R.B., Pedersen, C.N.S.: RNA pseudoknot prediction in energy based models. J. Comput. Biol. **7**, 409–428 (2000)
9. Rastegari, B., Condon, A.: Parsing nucleic acid pseudoknotted secondary structure: algorithm and applications. J. Comput. Biol. **14**(1), 16–32 (2007)
10. Reeder, J., Giegerich, R.: Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics. BMC Bioinform. **5**, 104 (2004)
11. Rivas, E., Eddy, S.: A dynamic programming algorithm for RNA structure prediction including pseudoknots. J. Mol. Biol. **285**, 2053–2068 (1999)
12. Rødland, E.A.: Pseudoknots in RNA secondary structure: Representation, enumeration, and prevalence. J. Comput. Biol. **13**, 1197–1213 (2006)
13. Tabaska, J.E., Cary, R.B., Gabow, H.N., Stormo, G.D.: An RNA folding method capable of identifying pseudoknots and base triples. Bioinform. **14**, 691–699 (1998)
14. Uemura, Y., Hasegawa, A., Kobayashi, S., Yokomori, T.: Tree adjoining grammars for RNA structure prediction. Theor. Comput. Sci. **210**, 277–303 (1999)
15. Witwer, C., Hofacker, I.L., Stadler, P.F.: Prediction of consensus RNA secondary structures including pseudoknots. IEEE Trans. Comput. Biol. Bioinform. **1**, 66–77 (2004)

RNA Secondary Structure Prediction by Minimum Free Energy

2006; Ogurtsov, Shabalina, Kondrashov, Roytberg

Rune B. Lyngsø
Department of Statistics, Oxford University, Oxford, UK

Keywords and Synonyms

RNA Folding

Problem Definition

This problem is concerned with predicting the set of base pairs formed in the native structure of an RNA molecule. The main motivation stems from structure being crucial for function and the growing appreciation of the importance of RNA molecules in biological processes. Base pairing is the single most important factor determining structure formation. Knowledge of the secondary structure alone also provides information about stretches of unpaired bases that are likely candidates for active sites. Early work [7] focused on finding structures maximizing the number of base pairs. With the work of Zuker and Stiegler [17] focus shifted to energy minimization in a model approximating the Gibbs free energy of structures.

Notation

Let $s \in \{A, C, G, U\}^*$ denote the sequence of bases of an RNA molecule. Use $X \cdot Y$ where $X, Y \in \{A, C, G, U\}$ to denote a base pair between bases of type X and Y, and $i \cdot j$ where $1 \leq i < j \leq |s|$ to denote a base pair between bases $s[i]$ and $s[j]$.

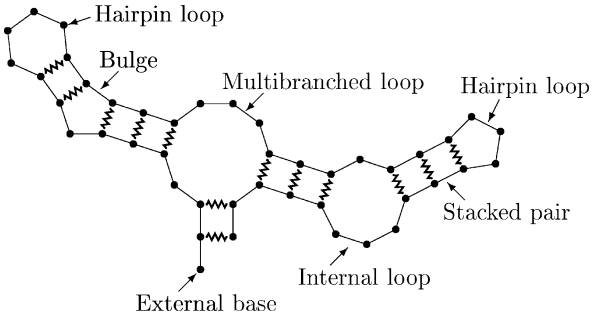
Definition 1 (RNA Secondary Structure) A secondary structure for an RNA sequence s is a set of base pairs $S = \{i \cdot j \mid 1 \leq i < j \leq |s| \wedge i < j - 3\}$. For $i \cdot j, i' \cdot j' \in S$ with $i \cdot j \neq i' \cdot j'$

- $\{i, j\} \cap \{i', j'\} = \emptyset$ (each base pairs with at most one other base)
- $\{s[i], s[j]\} \in \{\{A, U\}, \{C, G\}, \{G, U\}\}$ (only Watson-Crick and G, U wobble base pairs)
- $i < i' < j \Rightarrow j' < j$ (base pairs are either nested or juxtaposed but not overlapping).

The second requirement, that only canonical base pairs are allowed, is standard but not consequential in solutions to the problem. The third requirement states that the structure does not contain pseudoknots. This restriction is crucial for the results listed in this entry.

Energy Model

The model of Gibbs free energy applied, usually referred to as the nearest-neighbor model, was originally proposed by Tinoco et al. [10,11]. It approximates the free energy by postulating that the energy of the full three dimensional structure only depends on the secondary structure, and that this in turn can be broken into a sum of independent contributions from each loop in the secondary structure.



RNA Secondary Structure Prediction by Minimum Free Energy, Figure 1
A hypothetical RNA structure illustrating the different loop types. Bases are represented by *circles*, the RNA backbone by *straight lines*, and base pairs by *zigzagged lines*

Definition 2 (Loops) For $i \cdot j \in S$, base k is *accessible* from $i \cdot j$ iff $i < k < j$ and $\neg\exists i' \cdot j' \in S : i < i' < k < j' < j$. The *loop closed by* $i \cdot j$, $\ell_{i \cdot j}$, consists of $i \cdot j$ and all the bases accessible from $i \cdot j$. If $i' \cdot j' \in S$ and i' and j' are accessible from $i \cdot j$, then $i' \cdot j'$ is an interior base pair in the loop closed by $i \cdot j$.

Loops are classified by the number of interior base pairs they contain:

- hairpin loops have no interior base pairs
- stacked pairs, bulges, and internal loops have one interior base pair that is separated from the closing base pair on neither side, on one side, or on both sides, respectively
- multibranched loops have two or more interior base pairs.

Bases not accessible from any base pair are called external. This is illustrated in Fig. 1. The free energy of structure S is

$$'G(S) = \sum_{i \cdot j \in S} 'G(\ell_{i \cdot j}) \,, \tag{1}$$

where $'G(\ell_{i \cdot j})$ is the free energy contribution from the loop closed by $i \cdot j$.

Problem 1 (Minimum Free Energy Structure)
INPUT: *RNA sequence s and specification of $'G$ for all loops.*
OUTPUT: $\arg\min_S \{'G(S) \mid S$ *secondary structure for* $s\}$.

Key Results

Solutions are based on using dynamic programming to solve the general recursion

$$V[i,j] = \min_{k\geq 0; i<i_1<j_1<\ldots<i_k<j_k<j} \left\{ \Delta G(\ell_{i\cdot j; i_1\cdot j_1,\ldots,i_k\cdot j_k}) + \sum_{l=1}^{k} V[i_l, j_l] \right\}$$

$$W[i] = \min \left\{ W[i-1], \min_{0<k<i} \{W[k-1] + V[k,i]\} \right\} ,$$

where $'G(\ell_{i\cdot j; i_1\cdot j_1,\ldots,i_k\cdot j_k})$ is the free energy of the loop closed by $i \cdot j$ and interior base pairs $i_1 \cdot j_1, \ldots, i_k \cdot j_k$ and with initial condition $W[0] = 0$. In the following it is assumed that all loop energies can be computed in time $O(1)$.

Theorem 1 *If the free energy of multibranched loops is a sum of*

- *an affine function of the number of interior base pairs and unpaired bases*
- *contributions for each base pair from stacking with either neighboring unpaired bases in the loop or with a neighboring base pair in the loop, whichever is more favorable,*

a minimum free energy structure can be computed in time $O(|s|^4)$ *and space* $O(|s|^2)$ [17].

With these assumptions the time required to handle the multibranched loop parts of the recursion reduces to $O(|s|^3)$. Hence handling the $O(|s|^4)$ possible internal loops becomes the bottleneck.

Theorem 2 *If furthermore the free energy of internal loops is a sum of*

- *a function of the total size of the loop, i. e. the number of unpaired bases in the loop,*
- *a function of the asymmetry of the loop, i. e. the difference in number of unpaired bases on the two sides of the loop,*
- *contributions from the closing and interior base pairs stacking with the neighboring unpaired bases in the loop,*

a minimum free energy structure can be computed in time $O(|s|^3)$ *and space* $O(|s|^2)$ [5].

Under these assumptions the time required to handle internal loops reduces to $O(|s|^3)$. With further assumptions on the free energy contributions of internal loops this can be reduced even further, again making the handling of multibranched loops the bottleneck of the computation.

Theorem 3 *If furthermore the size dependency is concave and the asymmetry dependency is constant for all but* $O(1)$ *values, a multibranched loop free minimum free energy structure can be computed in time* $O(|s|^2 \log^2 |s|)$ *and space* $O(|s|^2)$ [8].

The above assumptions are all based on the nature of current loop energies [6]. These energies have to a large part been developed without consideration of computational expediency and parameters determined experimentally, although understanding of the precise behavior of larger loops is limited. For multibranched loops some theoretical considerations [4] would suggest that a logarithmic dependency would be more appropriate.

Theorem 4 *If the restriction on the dependency on number of interior base pairs and unpaired bases in Theorem 1 is weakened to any function that depends only on the number of interior base pairs, the number of unpaired bases, or the total number of bases in the loop, a minimum free energy structure can be computed in time* $O(n^4)$ *and space* $O(n^3)$ [13].

Theorem 5 *All the above theorems can be modified to compute a data structure that for any* $1 \leq i < j \leq |s|$ *allows us to compute the minimum free energy of any structure containing* $i \cdot j$ *in time* $O(1)$ [15].

Applications

Naturally the key application of these algorithms are for predicting the secondary structure of RNA molecules. This holds in particular for sequences with no homologues with common structure, e. g. functional analysis based on mutational effects and to some extent analysis of RNA aptamers. With access to structurally conserved homologues prediction accuracy is significantly improved by incorporating comparative information [2].

Incorporating comparative information seems to be crucial when using secondary structure prediction as the basis of RNA gene finding. As it turns out, the minimum free energy of known RNA genes is not sufficiently different from the minimum free energy of comparable random sequences to reliably separate the two [9,14]. However, minimum free energy calculations is at the core of one successful comparative RNA gene finder [12].

Open Problems

Most current research is focused on refinement of the energy parametrization. The limiting factor of sequence lengths for which secondary structure prediction by the methods described here is still feasible is adequacy of the nearest neighbor approximation rather than computation time and space. Still improvements on time and space

complexities are useful as biosequence analyzes are invariably used in genome scans. In particular improvements on Theorem 4, possibly for dependencies restricted to be logarithmic or concave, would allow for more advanced scoring of multibranched loops. A more esoteric open problem is to establish the complexity of computing the minimum free energy under the general formulation of (1), with no restrictions on loop energies except that they are computable in time polynomial in $|s|$.

Experimental Results

With the release of the most recent energy parameters [6] secondary structure prediction by finding a minimum free energy structure was found to recover approximately 73% of the base pairs in a benchmark data set of RNA sequences with known secondary structure. Another independent assessment [1] put the recovery percentage somewhat lower at around 56%. This discrepancy is discussed and explained in [1].

Data Sets

Families of homologous RNA sequences aligned and annotated with secondary structure are available from the Rfam data base at www.sanger.ac.uk/Software/Rfam/. Three dimensional structures are available from the Nucleic Acid Database at ndbserver.rutgers.edu/. An extensive list of this and other data bases is available at www.imb-jena.de/RNA.html.

URL to Code

Software for RNA folding and a range of related problems is available from www.bioinfo.rpi.edu/applications/hybrid/download.php and www.tbi.univie.ac.at/~ivo/RNA/. Software implementing the efficient handling of internal loops of [8] is available from ftp.ncbi.nlm.nih.gov/pub/ogurtsov/Afold.

Cross References

- ► RNA Secondary Structure Boltzmann Distribution
- ► RNA Secondary Structure Prediction Including Pseudoknots

Recommended Reading

1. Dowell, R., Eddy, S.R.: Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. BMC Bioinformatics **5**, 71 (2004)
2. Gardner, P.P., Giegerich, R.: A comprehensive comparison of comparative RNA structure prediction approaches. BMC Bioinformatics **30**, 140 (2004)
3. Hofacker, I.L., Stadler, P.F.: Memory efficient folding algorithms for circular RNA secondary structures. Bioinformatics **22**, 1172–1176 (2006)
4. Jacobson, H., Stockmayer, W.H.: Intramolecular reaction in polycondensations. I. the theory of linear systems. J. Chem. Phys. **18**, 1600–1606 (1950)
5. Lyngsø, R.B., Zuker, M., Pedersen, C.N.S., Fast evaluation of internal loops in RNA secondary structure prediction. Bioinformatics **15**, 440–445 (1999)
6. Mathews, D.H., Sabina, J., Zuker, M., Turner, D.H.: Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. J. Mol. Biol. **288**, 911–940 (1999)
7. Nussinov, R., Jacobson, A.B.: Fast algorithm for predicting the secondary structure of single-stranded RNA. Proc. Natl. Acad. Sci. USA **77**, 6309–6313 (1980)
8. Ogurtsov, A.Y., Shabalina, S.A., Kondrashov, A.S., Roytberg, M.A.: Analysis of internal loops within the RNA secondary structure in almost quadratic time. Bioinformatics **22**, 1317–1324 (2006)
9. Rivas, E., Eddy, S.R.: Secondary structure alone is generally not statistically significant for the detection of noncoding RNAs. Bioinformatics **16**, 583–605 (2000)
10. Tinoco, I., Borer, P.N., Dengler, B., Levine, M.D., Uhlenbeck, O.C., Crothers, D.M., Gralla, J.: Improved estimation of secondary structure in ribonucleic acids. Nat. New Biol. **246**, 40–41 (1973)
11. Tinoco, I., Uhlenbeck, O.C., Levine, M.D.: Estimation of secondary structure in ribonucleic acids. Nature **230**, 362–367 (1971)
12. Washietl, S., Hofacker, I.L., Stadler, P.F.: Fast and reliable prediction of noncoding RNA. Proc. Natl. Acad. Sci. USA **102**, 2454–59 (2005)
13. Waterman, M.S., Smith, T.F.: Rapid dynamic programming methods for RNA secondary structure. Adv. Appl. Math. **7**, 455–464 (1986)
14. Workman, C., Krogh, A.: No evidence that mRNAs have lower folding free energies than random sequences with the same dinucleotide distribution. Nucleic Acids Res. **27**, 4816–4822 (1999)
15. Zuker, M.: On finding all suboptimal foldings of an RNA molecule. Science **244**, 48–52 (1989)
16. Zuker, M.: Calculating nucleic acid secondary structure. Curr. Opin. Struct. Biol. **10**, 303–310 (2000)
17. Zuker, M., Stiegler, P.: Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. Nucleic Acids Res. **9**, 133–148 (1981)

Robotics

1997; (Navigation) Blum, Raghavan, Schieber
1998; (Exploration) Deng, Kameda, Papadimitriou
2001; (Localization) Fleischer, Romanik, Schuierer, Trippen

Rudolf Fleischer
Deptartment of Computer Science and Engineering, Fudan University, Shanghai, China

Keywords and Synonyms

Navigation problem – Search problem
Exploration problem – Mapping problem; Gallery tour problem
Localization problem – Kidnapped robot problem

Problem Definition

Definitions

There are three fundamental algorithmic problems in robotics: exploration, navigation, and localization. *Exploration* means to draw a complete map of an unknown environment. *Navigation* (or *search*) means to find a way to a predescribed location among unknown obstacles. *Localization* means to determine the current position on a known map. Normally, the environment is modeled as a simple polygon with or without holes. To distinguish the underlying combinatorial problems from the geometric problems, the environment may also be modeled as a graph.

Normally, a robot has a compass, i. e., it can distinguish between different directions, and it can measure travel distance. A *blind* (or *tactile*) robot can only sense its immediate surroundings (for example, it only notices an obstacle when it bumps into it; this is also sometimes called ε-*radar*), while a robot *with vision* can see objects far in the distance, unless the view is blocked by opaque obstacles. Robots on graphs are usually blind. In polygonal environments, vision may help to judge the size of an obstacle without moving, but a blind robot can circumvent obstacles with a performance loss of only a factor of nine by using the *lost-cow* doubling strategy [2].

Online Algorithms

An algorithm that tries to approximate an optimal solution by making decisions under a given uncertainty is called an *online algorithm* (see the surveys in [9]). Its performance is measured by the *competitive ratio*, which is the approximation ratio of the online algorithm maximized over all possible input scenarios. In the case of exploration, navigation, and localization, the robot should minimize its travel distance. Therefore, the competitive ratio measures the length of the detour compared to the optimal shortest tour.

A *randomized* online algorithm against an *oblivious adversary* uses randomization on a fixed predetermined input (which is unknown to the online algorithm). In this case, the competitive ratio is a random variable, and it is maximized over all possible inputs.

Exploration

Deng et. al [7] introduced the *gallery tour problem*. Given a polygonal room with polygonal obstacles, an *entry point s* and *exit point t*, a robot with vision needs to travel along a path from s to t such that it can see every point of the perimeter of the polygons. If $s = t$, the problem is known as the *watchman's route problem*. In the online version of the problem, the polygon is initially unknown. The problem becomes easier in rectilinear environments with L_1-metric.

Navigation

Blum et. al [5] studied the problem of a blind robot trying to reach a goal t from a start position s (*point-to-point navigation*) in a scene of non-overlapping axis-parallel rectangles of width at least one. In the *wall problem*, t is an infinite vertical line. In the *room problem*, the obstacles are within a square room with entry door s.

Localization

In the localization problem the robot knows a map of the environment, but not its current position, which it determines by moving around and matching the observed local environment with the given map.

Key Results

Exploration

Theorem 1 ([7]) *The shortest exploration tour in L_1-metric in a known simple rectilinear polygon with n vertices can be computed in time $O(n^3)$.*

Theorem 2 ([15]) *There is a 26.5-competitive online algorithm to explore an unknown simple polygon without obstacles.*

Theorem 3 ([7]) *There is an $O(k+1)$-competitive online algorithm to explore an unknown simple polygon with k polygonal obstacles.*

Theorem 4 ([1]) *No randomized online algorithm can explore an unknown simple rectilinear polygon with k rectilinear obstacles better than $\Omega(\sqrt{k})$-competitively.*

Navigation

Theorem 5 ([19]) *No online algorithm for the wall problem with n rectangles can be better than $\Omega(\sqrt{n})$-competitive.*

Theorem 6 ([5]) *There are $O(\sqrt{n})$-competitive online algorithms for the wall problem, the room problem, and point-to-point navigation in scenes with n axis-parallel rectangles.*

Theorem 7 ([3]) *There is an optimal $\Theta(\log n)$-competitive online algorithms for the room problem with n axis-parallel rectangles.*

Theorem 8 ([4]) *There are $O(\log n)$-competitive randomized online algorithms against oblivious adversary for the wall problem and point-to-point navigation in scenes with n axis-parallel rectangles.*

Theorem 9 ([16]) *No randomized online algorithm against oblivious adversary for point-to-point navigation between n rectangles can be better than $\Omega(\log \log n)$-competitive.*

Theorem 10 ([5]) *There is a lower bound of $n/8$ for the competitiveness of navigating between n non-convex obstacles. A simple memoryless algorithm achieves a competitive ratio of $50.4 \cdot n$.*

Localization

Theorem 11 ([17]) *No algorithm for localization in geometric trees with n nodes can be better than $\Omega(\sqrt{n})$-competitive.*

Theorem 12 ([11]) *There is an $O(\sqrt{n})$-competitive algorithm for localization in geometric trees with n nodes.*

Applications

Exploration

It is NP-hard to find a shortest exploration tour in a known polygonal environment [7]. Unknown scenes with arbitrary obstacles can be efficiently explored by Lumelsky's `Sightseer Strategy`. Most online exploration algorithms can be transformed into an efficient online algorithm to approximate the *search ratio*, a measure related to the competitive ratio of the navigation problem [10].

The problem of exploring a polygonal environment is closely related to the problem of exploring strongly connected digraphs. Here, the competitive ratio is usually given as a function of the *deficiency* of the graph which is the minimum number of edges that must be added to the graph to make it Eulerian. Eulerian graphs can be explored with a simple optimal 2-competitive algorithm, while graphs of deficiency d can be explored with a competitive factor of $O(d^8)$ [14].

Navigation

In applied robotics, it is common to measure the competitive ratio as a function of the aspect ratio of the obstacle rectangles. Lumelsky's `BUG2` algorithm can navigate between convex obstacles, in the worst case moving at most once around every obstacle, which is optimal.

A robot with a compass can sometimes find the goal exponentially faster than a robot without a compass.

If we need to do several trips in an unknown environment, it may help to use partial map information from previous trips. In particular, the i-th trip between the same two points can be searched $\sqrt{\frac{n}{i}}$-competitively.

Localization

There is a simple k-competitive localization algorithm in polygons and graphs, where k is the number of positions on the map matching the observed environment at the wake-up point.

The *visibility polygon* of a point v is that part of a polygon that a robot can see when sitting at v. One can compute in polynomial time all points of a given simple polygon whose visibility polygon matches a given star polygon.

Computing a shortest localization tour in a known polygon is NP-hard. It can be approximated with a factor of $O(\log^3 n)$, but not better than $\Omega(\log n)$ unless $P = NP$ [18].

Open Problems

Exploration

- A polynomial time algorithm for computing the shortest exploration tour in a known simple polygon without obstacles. Such an algorithm is known for the watchman's route problem.
- A simple online exploration algorithm for simple polygons with tight analysis.
- Exploration and navigation with limited memory.

Navigation

- Online searching among convex polygonal obstacles.
- Three-dimensional navigation, in particular among non-convex obstacles (3d-mazes).

Localization

- A simple algorithm for online localization in trees.
- Online localization in general graphs.
- A randomized online localization algorithm beating deterministic algorithms.

Experimental Results

Exploration

Fleischer and Trippen [13] implemented most known algorithms for exploring a directed graph and demonstrated that the simple (but inferior) greedy algorithms usually outperform the more sophisticated algorithms on random graphs.

Navigation

Coffman and Gilbert [6] implemented eight heuristics for point-to-point navigation in L_1-metric.

Localization

Fleischer and Trippen [12] visualized their localization algorithm in geometric trees.

Cross References

- ► Alternative Performance Measures in Online Algorithms
- ► Deterministic Searching on the Line
- ► Metrical Task Systems
- ► Randomized Searching on Rays or the Line

Recommended Reading

1. Albers, S., Kursawe, K., Schuierer, S.: Exploring unknown environments with obstacles. Algorithmica **32**(1), 123–143 (2002)
2. Baeza-Yates, R.A., Culberson, J.C., Rawlins, G.J.E.: Searching in the plane. Inf. Comput. **106**(2), 234–252 (1993)
3. Bar-Eli, E., Berman, P., Fiat, A., Yan, P.: Online navigation in a room. J. Algorithms **17**(3), 319–341 (1994)
4. Berman, P., Blum, A., Fiat, A., Karloff, H., Rosén, A., Saks, M.: Randomized robot navigation algorithms. In: Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms (SODA'96), 1996, pp. 75–84
5. Blum, A., Raghavan, P., Schieber, B.: Navigating in unfamiliar geometric terrain. SIAM J. Comput. **26**(1), 110–137 (1997)
6. Coffman Jr., E.G., Gilbert, E.N.: Paths through a maze of rectangles. Networks **22**, 349–367 (1992)
7. Deng, X., Kameda, T., Papadimitriou, C.H.: How to learn an unknown environment. J. ACM **45**, 215–245 (1998)
8. Dudek, G., Romanik, K., Whitesides, S.: Localizing a robot with minimum travel. SIAM J. Comput. **27**(2), 583–604 (1998)
9. Fiat, A., Woeginger, G. (eds.) Online Algorithms – The State of the Art. Springer Lecture Notes in Computer Science, vol. 1442. Springer, Heidelberg (1998)
10. Fleischer, R., Kamphans, T., Klein, R., Langetepe, E., Trippen, G.: Competitive online approximation of the optimal search ratio. In: Proceedings of the 12th European Symposium on Algorithms (ESA'04). Lecture Notes in Computer Science, vol. 3221, pp. 335–346. Springer, Heidelberg (2004)
11. Fleischer, R., Romanik, K., Schuierer, S., Trippen, G.: Optimal robot localization in trees. Inf. Comput. **171**, 224–247 (2001)
12. Fleischer, R., Trippen, G.: Optimal robot localization in trees. In: Proceedings of the 16th Annual Symposium on Computational Geometry (SoCG'00), 2000, pp. 373–374. A video shown at the 9th Annual Video Review of Computational Geometry
13. Fleischer, R., Trippen, G.: Experimental studies of graph traversal algorithms. In: Proceedings of the 2nd International Workshop on Experimental and Efficient Algorithms (WEA'03). Lecture Notes in Computer Science, vol. 2647, pp. 120–133. Springer, Heidelberg (2003)
14. Fleischer, R., Trippen, G.: Exploring an unknown graph efficiently. In: Proceedings of the 13th European Symposium on Algorithms (ESA'05). Lecture Notes in Computer Science, vol. 3669, pp. 11–22. Springer, Heidelberg (2005)
15. Hoffmann, F., Icking, C., Klein, R., Kriegel, K.: The polygon exploration problem. SIAM J. Comput. **31**(2), 577–600 (2001)
16. Karloff, H., Rabani, Y., Ravid, Y.: Lower bounds for randomized *k*-server and motion-planning algorithms. SIAM J. Comput. **23**(2), 293–312 (1994)
17. Kleinberg, J.M.: The localization problem for mobile robots. In: Proceedings of the 35th Symposium on Foundations of Computer Science (FOCS'94), 1994, pp. 521–531
18. Koenig, S., Mudgal, A., Tovey, C.: A near-tight approximation lower bound and algorithm for the kidnapped robot problem. In: Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA'06), 2006, pp. 133–142.
19. Papadimitriou, C.H., Yannakakis, M.: Shortest paths without a map. Theor. Comput. Sci. **84**, 127–150 (1991)

Robust Geometric Computation

2004; Li, Yap

Chee K. Yap, Vikram Sharma
Department of Computer Science, New York University, New York, NY, USA

Keywords and Synonyms

Exact geometric computation Floating-point filter; Dynamic and static filters; Topological consistency

Problem Definition

Algorithms in computational geometry are usually designed under the Real RAM model. In implementing these algorithms, however, fixed-precision arithmetic is used in place of exact arithmetic. This substitution introduces numerical errors in the computations that may lead to nonrobust behavior in the implementation, such as infinite loops or segmentation faults.

There are various approaches in the the literature addressing the problem of nonrobustness in geometric computations; see [9] for a survey. These approaches can be classified along two lines: the **arithmetic approach** and the **geometric approach**.

The arithmetic approach tries to address nonrobustness in geometric algorithms by handling the numerical errors arising because of fixed-precision arithmetic; this can be done, for instance, by using multi-precision arithmetic [6], or by using rational arithmetic whenever possible. In general, all the arithmetic operations, including exact comparison, can be performed on algebraic quantities. The drawback of such a general approach is its inefficiency.

The geometric approaches guarantee that certain geometric properties are maintained by the algorithm. For example, if the Voronoi diagram of a planar point set is being computed then it is desirable to ensure that the output is a planar graph as well. Other geometric approaches are finite resolution geometry [7], approximate predicates and fat geometry [8], consistency and topological approaches [4], and topology oriented approach [13]. The common drawback of these approaches is that they are problem or algorithm specific.

In the past decade, a general approach called the **Exact Geometric Computation** (EGC) [15] has become very successful in handling the issue of nonrobustness in geometric computations; strictly speaking, this approach is subsumed in the arithmetic approaches. To understand the EGC approach, it helps to understand the two parts common to all geometric computations: a *combinatorial structure* characterizing the discrete relations between geometric objects, e. g., whether a point is on a hyperplane or not; and a *numerical part* that consists of the numerical representation of the geometric objects, e. g. the coordinates of a point expressed as rational or floating-point numbers. Geometric algorithms characterize the combinatorial structure by numerically computing the discrete relations (that are embodied in geometric predicates) between geometric objects. Nonrobustness arises when numerical errors in the computations yield an incorrect characterization. The EGC approach ensures that all the geometric predicates are evaluated correctly thereby ensuring the correctness of the computed combinatorial structure and hence the robustness of the algorithm.

Notation

An **expression** E refers to a syntactic object constructed from a given set of operators over the reals $\mathbb{R}$. For example, the set of expressions on the set of operators $\{\mathbb{Z}, +, -, \times, \sqrt{}\}$ is the set of division-free radical expressions on the integers; more concretely, expressions can be viewed as directed acyclic graphs (DAG) where the internal nodes are operators with arity at least one, and the leaves are constants, i. e., operators with arity zero. The value of an expression is naturally defined using induction; note that the value may be undefined. Let E represent both the value of the expression and the expression itself.

Key Results

Following are the key results that have led to the feasibility and success of the EGC approach.

Constructive Zero Bounds

The possibility of EGC approach hinges on the computability of the sign of an expression. For determining the sign of algebraic expressions EGC libraries currently use a numerical approach based upon zero bounds. A **zero bound** $b > 0$ for an expression E is such that absolute value $|E|$ of E is greater than b if the value of E is valid and nonzero. To determine the sign of the expression E, compute an approximation $\tilde{E}$ to E such that $|\tilde{E} - E| < \frac{b}{2}$ if E is valid, otherwise $\tilde{E}$ is also invalid. Then sign of E is the same as the sign of $\tilde{E}$ if $|\tilde{E}| \geq \frac{b}{2}$, otherwise it is zero. A **constructive zero bound** is an effectively computable function B from the set of expressions to real numbers $\mathbb{R}$ such that $B(E)$ is a zero bound for any expression E. For examples of constructive zero bounds, see [2,11].

Approximate Expression Evaluation

Another crucial feature in developing the EGC approach is developing algorithms for approximate expression evaluation, i. e., given an expression E and a relative or absolute precision p, compute an approximation to the value of the expression within precision p. The main computational paradigm for such algorithms is the **precision-driven approach** [15]. Intuitively, this is a downward-upward process on the input expression DAG; propagate precision values down to the leaves in the downward direction; at the leaves of the DAG, assume the ability to approximate the value associated with the leaf to any desired precision; finally, propagate the approximations in the upward direction towards the root. Ouchi [10] has given detailed algorithms for the propagation of "composite precision", a generalization of relative and absolute precision.

Numerical Filters

Implementing approximate expression evaluation requires multi-precision arithmetic. But efficiency can be gained by exploiting machine floating-point arithmetic, which is fast and optimized on current hardware. The basic idea is to to check the output of machine evaluation of predicates, and fallback on multi-precision methods if the check fails. These checks are called numerical

filters; they certify certain properties of computed numerical values, such as their sign. There are two main classifications of numerical filters: *static filters* are those that can be mostly computed at compile time, but they yield overly pessimistic error bounds and thus are less effective; *dynamic filters* are implemented during run time and even though they have higher costs they are much more effective than static filters, i. e., have better estimate on error bounds. See Fortune and van Wyk [5].

Applications

The EGC approach has led to the development of libraries, such as LEDA Real and CORE, that provide EGC number types, i. e., a class of expressions whose signs are guaranteed. CGAL, another major EGC Library that provides robust implementation of algorithms in computational geometry, offers various specialized EGC number types, but for general algebraic numbers it can also use LEDA Real or CORE.

Open Problems

1. An important challenge from the perspective of efficiency for EGC approach is high degree algebraic computation, such as those found in Computer Aided Design. These issues are beginning to be addressed, for instance [1].
2. The *fundamental problem of EGC* is the **zero problem**: given any set of real algebraic operators, decide whether any expression over this set is zero or not. The main focus here is on the decidability of the zero problem for non-algebraic expressions. The importance of this problem has been highlighted by Richardson [12]; recently some progress has been made for special non-algebraic problems [3].
3. When algorithms in EGC approach are embedded in larger application systems (such as mesh generation systems), the output of one algorithm needs to be cascaded as input to another; the output of such algorithms may be in high precision, so it is desirable to reduce the precision in the cascade. The geometric version of this problem is called the **geometric rounding problem**: given a consistent geometric object in high precision, "round" it to a consistent geometric object at a lower precision.
4. Recently a computational model for the EGC approach has been proposed [14]. The corresponding complexity model needs to be developed. Standard complexity analysis based on input size is inadequate for evaluating the complexity of real computation; the complexity should be expressed in terms of the output precision.

URL to Code

1 `Core Library`: http://www.cs.nyu.edu/exact
2 `LEDA`: http://www.mpi-sb.mpg.de/LEDA
3 `CGAL`: http://www.cgal.org

Recommended Reading

1. Berberich, E., Eigenwillig, A., Hemmer, M., Hert, S., Schmer, K. M., Schmer, E.: A computational basis for conic arcs and boolean operations on conic polygons. In: 10th European Symposium on Algorithms (ESA'02), pp. 174–186, (2002) Lecture Notes in CS, No. 2461
2. Burnikel, C., Funke, S., Mehlhorn, K., Schirra, S., Schmitt, S.: A separation bound for real algebraic expressions. In: Lecture Notes in Computer Science, pp. 254–265. Springer, vol 2161 (2001)
3. Chang, E.C., Choi, S.W., Kwon, D., Park, H., Yap, C.: Shortest Paths for Disc Obstacles is Computable. In: Gao, X.S., Michelucci, D. (eds.) Special Issue on Geometric Constraints. Int. J. Comput. Geom. Appl. **16**(5–6), 567–590 (2006), Also appeared in Proc. 21st ACM Symp. Comp. Geom., pp. 116–125 (2005)
4. Fortune, S.J.: Stable maintenance of point-set triangulations in two dimensions. IEEE Found. Comput. Sci.: **30**, 494–499 (1989)
5. Fortune, S.J., van Wyk, C.J.: Efficient exact arithmetic for computational geometry. In: Proceeding 9th ACM Symposium on Computational Geometry, pp. 163–172 (1993)
6. Gowland, P., Lester, D.: Asurvey of exact arithmetic implementations. In: Blank, J., Brattka, V., Hertling, P. (eds.) Computability and Complexity in Analysis, pp. 30–47. Springer, 4th International Workshop, CCA 2000, Swansea, UK, September 17–19, (2000), Selected Papers. Lecture Notes in Computer Science, No. 2064
7. Greene, D.H., Yao, F.F.: Finite-resolution computational geometry. IEEE Found. Comput. Sci. **27**, 143–152 (1986)
8. Guibas, L., Salesin, D., Stolfi, J.: Epsilon geometry: building robust algorithms from imprecise computations. ACM Symp Comput. Geometr. **5**, 208–217 (1989)
9. Li, C., Pion, S., Yap, C.K.: Recent progress in Exact Geometric Computation. J. Log. Algebr. Program. **64**(1), 85–111 (2004)
10. Ouchi, K.: Real/Expr: Implementation of an exact computation package. Master's thesis, New York University, Department of Computer Science, Courant Institute, January (1997). URL http://cs.nyu.edu/exact/doc/
11. Pion, S., Yap, C.: Constructive root bound method for k-ary rational input numbers, September, (2002). Extended Abstract. Submitted, (2003) ACM Symposium on Computational Geometry
12. Richardson, D.: How to recognize zero. J. Symb. Comput. **24**, 627–645 (1997)
13. Sugihara, K., Iri, M., Inagaki, H., Imai, T.: Topology-oriented implementation—an approach to robust geometric algorithms. Algorithmica **27**, 5–20 (2000)
14. Yap, C.K.: Theory of Real Computation according to EGC. To appear in LNCS Volume based on talks at a Dagstuhl Seminar "Reliable Implementation of Real Number Algorithms: Theory and Practice", Jan 8–13, (2006)
15. Yap, C.K., Dubé, T.: The exact computation paradigm. In: Du, D.Z., Hwang, F.K.: (eds.) Computing in Euclidean Geometry, 2nd edn., pp. 452–492. World Scientific Press, Singapore (1995)

Robustness

- ▶ Connectivity and Fault-Tolerance in Random Regular Graphs
- ▶ Distance-Based Phylogeny Reconstruction (Optimal Radius)
- ▶ False-Name-Proof Auction

Routing

2003; Azar, Cohen, Fiat, Kaplan, Räcke

József Békési, Gábor Galambos
Department of Computer Science, Juhász Gyula Teachers Training College, Szeged, Hungary

Keywords and Synonyms

Routing algorithms; Network flows; Oblivious routing

Problem Definition

One of the most often used techniques in modern computer networks is routing. *Routing* means selecting paths in a network along which to send data. Demands usually randomly appear on the nodes of a network, and routing algorithms should be able to send data to their destination. The transfer is done through intermediate nodes, using the connecting links, based on the topology of the network. The user waits for the network to guarantee that it has the required capacity during data transfer, meaning that the network behaves like its nodes would be connected directly by a physical line. Such service is usually called the *permanent virtual circuit (PVC)* service. To model real life situations, assume that demands arrive *on line*, given by source and destination points, and capacity (bandwidth) requirements.

Similar routing problems may occur in other environments, for example in parallel computation. In this case there are several processors connected together by wires. During an operation some data appear at given processors which should be sent to specific destinations. Thus, this also defines a routing problem. However, this paper mainly considers the network model, not the parallel computer one.

For any given situation there are several routing possibilities. A natural question is to ask which is the best possible algorithm. To find the best algorithm one must define an objective function, which expresses the effectiveness of the algorithm. For example, the aim may be to reduce the load of the network. Load can be measured in different ways, but to measure the utilization percent of the nodes or the links of the network is the most natural. In the online setting, it is interesting to compare the behavior of a routing algorithm designed for a specific instance to the best possible routing.

There are two fundamental approaches towards routing algorithms. The first approach is to route *adaptively*, i. e. depending on the actual loads of the nodes or the links. The second approach is to route *obviously*, without using any information about the current state of the network. Here the authors survey only results on oblivious routing algorithms.

Notations and Definitions

A mathematical model of the network routing problem is now presented.

Let $G(V, E, c)$ be a capacitated network, where V is the set of nodes and E is the set of edges with a capacity function $c : E \to R^+$. Let $|V| = n, |E| = m$. It can be assumed that G is directed, because if G is undirected then for each undirected edge $e = (u, v)$ two new nodes x, y and four new directed edges $e_1 = (u, x), e_2 = (v, x), e_3 = (y, u)$, $e_4 = (y, u)$ with infinite capacity may be added to the graph. If e is considered as an undirected edge with the same capacity then a directed network equivalent to the original one is received.

Definition 1 A set of functions $f := \{f_{ij} | i, j \in V, f_{ij} : E(G) \to R^+\}$ is called a **multi–commodity flow** if

$$\sum_{e \in E_k^+} f_{ij}(e) = \sum_{e \in E_k^-} f_{ij}(e)$$

holds for all $k \neq i, k \neq j$, where $k \in V$ and E_k^+, E_k^- are the set of edges coming out from k and coming into k resp. Each function f_{ij} defines a **single–commodity flow** from i to j.

Definition 2 The **value** of a multi–commodity *flow* is an $n \times n$ matrix $T_f = (t_{ij}^f)$, where

$$t_{ij}^f = \sum_{e \in E_i^+} f_{ij}(e) - \sum_{e \in E_i^-} f_{ij}(e)\,,$$

$$\text{if } i \neq j \text{ and } v_{ii}^f = 0\,, \text{ for all } i, j \in V\,.$$

Definition 3 Let D be a nonnegative $n \times n$ matrix where the diagonal entries are 0. D is called as **demand matrix**. The **flow** on an edge $e \in E$ routing the demand matrix D by routing r is defined by

$$\text{flow}(e, r, D) = \sum_{i,j \in V} d_{ij} r_{ij}(e)\,,$$

while the **edge congestion** is

$$\mathrm{con}(e, r, D) = \frac{\mathrm{flow}(e, r, D)}{c(e)} .$$

The **congestion** of demand D using routing r is

$$\mathrm{con}(r, D) = \max_{e \in E} \mathrm{con}(e, r, D) .$$

Definition 4 A multi–commodity flow r is called **routing** if $t^r_{ij} = 1$, and if $i \neq j$ for all $i, j \in V$.

Routing represents a way of sending information over a network. The real load of the edges can be represented by scaling the edge congestions with the demands.

Definition 5 The **oblivious performance ratio** P_r of routing r is

$$P_r = \sup_D \left\{ \frac{\mathrm{con}(r, D)}{\mathrm{opt}(D)} \right\}$$

where $opt(D)$ is the optimal congestion which can be achieved on D. The **optimal oblivious routing ratio** for a network G is denoted by $opt(G)$, where

$$\mathrm{opt}(G) = \min_r P_r$$

Problem
INPUT: *A capacitated network* $G(V, E, c)$.
OUTPUT: *An oblivious routing r, where* P_r *is minimal.*

Key Results

Theorem 1 *There is a polynomial time algorithm that for any input network G (directed or undirected) finds the optimal oblivious routing ratio and the corresponding routing r.*

Theorem 2 *There is a directed graph G of n vertices such that opt(G) is at least* $\Omega(\sqrt{n})$.

Applications

Most importantly, with these results one can efficiently calculate the best routing strategy for a network topology with capacity constraints. This is a good tool for network planning. The effectiveness of a given topology can be tested without any knowledge of the the network traffic using this analysis.

Many researchers have investigated the variants of routing problems. For surveys on the most important models and results, see [10] and [11]. Oblivious routing algorithms were first analyzed by Valiant and Brebner ([15]). Here, they considered the parallel computer model and investigated specific architectures, like hypercube, square grids, etc. Borodin and Hopcroft investigated general networks ([6]). They showed that such simple deterministic strategies like oblivious routing can not be very efficient for online routing and proved a lower bound on the competitive ration of oblivious algorithms. This lower bound was later improved by Kaklamanis et al. ([9]), and they also gave an optimal oblivious deterministic algorithm for the hypercube.

In 2002, Räcke constructed a polylog competitive randomized algorithm for general undirected networks. More precisely, he proved that for any demand there is a routing such that the maximum edge congestion is at most polylog(n) times the optimal congestion for this demand ([12]). The work of Azar et al. extends this result by giving a polynomial method for calculating the optimal oblivious routing for a network. They also prove that for directed networks no logarithmic oblivious performance ratio exists. Recently, Hajiaghayi et al. present an oblivious routing algorithm which is $O\left(\log^2 n\right)$-competitive with high probability in directed networks ([8]).

A special online model has been investigated in [5], where the authors define the so called "repeated game" setting, where the algorithm is allowed to chose a new routing in each day. This means that it is oblivious to the demands, that will occur the next day. They present an $1 + \varepsilon$-competitive algorithm for this model.

There are better algorithms for the adaptive case, for example in [2]. For the offline case Raghavan and Thomson gave an efficient algorithm in [13].

Open Problems

The authors investigated edge congestion in this paper, but in practice, node congestion may be interesting as well. Node congestion means the ratio of the total traffic traversing a node to its capacity. Some results can be found for this problem in [7] and in [3]. It is an open problem whether this method used for edge congestion analysis can be applied for such a model. Another interesting open question may be whether there is a more efficient algorithm to compute the optimal oblivious performance ratio of a network ([1,14]).

Experimental Results

The authors applied their method on ISP network topologies and found that the calculated optimal oblivious ratios are surprisingly low, between 1.4 and 2. Other research dealing with this question found similar results ([1,14]).

Cross References

- ▶ Approximate Maximum Flow Construction
- ▶ Direct Routing Algorithms
- ▶ Load Balancing
- ▶ Mobile Agents and Exploration
- ▶ Oblivious Routing
- ▶ Probabilistic Data Forwarding in Wireless Sensor Networks

Recommended Reading

1. Applegate, D., Cohen, E.: Making routing robust to changing traffic demands: algorithms and evaluation. IEEE/ACM Trans Netw **14**(6), 1193–1206 (2006). doi:10.1109/TNET.2006.886296
2. Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., Waarts, O.: On-line routing of virtual circuits with applications to load balancing and machine scheduling. J. ACM **44**(3), 486–504 (1997)
3. Azar, Y., Chaiutin, Y.: Optimal node routing. In: Proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science, 2006, pp. 596–607
4. Azar, Y., Cohen, E., Fiat, A., Kaplan, H. Räcke, H.: Optimal oblivious routing in polynomial time. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing, 2003, pp. 383–388
5. Bansal, N., Blum, A., Chawla, S.: Meyerson, A.: Online oblivious routing. In: Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms, 2003, pp. 44–49
6. Borodin, A., Hopcroft, J.E.: Routing, merging and sorting on parallel models of computation. J. Comput. Syst. Sci. **30**(1), 130–145 (1985)
7. Hajiaghayi, M.T., Kleinberg, R.D., Leighton, T., Räcke, H.: Oblivious routing on node-capacitated and directed graphs. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, 2005, pp. 782–790
8. Hajiaghayi, M.T., Kim, J.H., Leighton, T., Räcke, H.: Oblivious routing in directed graphs with random demands. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005, pp. 193–201
9. Kaklamanis, C., Krizanc, D., Tsantilas, A.: Tight bounds for oblivious routing in the hypercube. In: Proc. 2nd Annual ACM Symposium on Parallel Algorithms and Architectures, 1990, pp. 31–36
10. Leighton, F.T.: Introduction to Parallel Algorithms and Architectures Arrays, Trees, Hypercubes. Morgan Kaufmann Publishers, San Fransisco (1992)
11. Leonardi, S.: On-line network routing. In: Fiat, A., Woeginger, G. (eds.) Online Algorithms – The State of the Art. Chap. 11, pp. 242–267. Springer, Heidelberg (1998)
12. Räcke, H.: Minimizing Congestions in General Networks. In: Proceedings of the 43rd Symposium on Foundations of Computer Science, 2002, pp. 43–52
13. Raghavan, P., Thompson, C.D.: Randomized rounding: a technique for provably good algorithms and algorithmic proofs. Combinatorica **7**, 365–374 (1987)
14. Spring, N., Mahajan, R., Wetherall, D.: Measuring ISP topologies with Rocketfuel. In: Proceedings of the ACM SIGCOMM'02 Conference. ACM, New York (2002)
15. Valiant, L.G., Brebner, G.: Universal schemes for parallel communication. In: Proceedings of the 13th ACM Symposium on Theory of Computing, 1981, pp. 263–277

Routing in Geometric Networks

2003; Kuhn, Wattenhofer, Zhang, Zollinger

Leszek Gąsieniec, Chang Su, Prudence Wong
Department of Computer Science, University of Liverpool, Liverpool, UK

Keywords and Synonyms

Geometric routing; Geographic routing; Location-based routing

Problem Definition

Network Model/Communication Protocol

In geometric networks, the nodes are embedded into Euclidean plane. Each node is aware of its geographic location, i. e., it knows its (x, y) coordinates in the plane.

Each node has the same transmission range, i. e., if a node v is within the transmission range of another node u, the node u can transmit to v directly and vice versa. Thus, the network can be modeled as an undirected graph $G = (V, E)$, where two nodes $u, v \in V$ are connected by an edge $(u, v) \in E$ if u and v are within their transmission ranges. Such two nodes are called *neighboring nodes* or simply *neighbors*. If two nodes are outside of their transmission ranges a multi-hop transmission is involved, i. e., the two nodes must communicate via intermediate nodes.

The cost $c(e)$ of sending a message over an edge $e \in E$ to a neighboring node has been modeled in many different ways. The most common ones include: the *hop (link) metric* ($c(e) = 1$), the *Euclidean metric* ($c(e) = |e|$), where $|e|$ is the Euclidean length of the edge e, and the *energy metric* ($c(e) = |e|^{\alpha}$ for $\alpha \geq 2$).

In geometric networks there is no fixed infrastructure nor a central server. I.e., all the nodes act as hosts as well as routers. The topology of the network is unknown to the nodes apart from their direct neighborhood, i. e., each node is aware of its own location as well as the coordinates of its neighbors. The nodes need to discover and maintain routes (involved in multi-hop transmissions) by themselves in a distributed manner. It is also very often assumed (in the context of sensor networks) that each node has limited memory and power.

Geometric routing is to route a message from a *source node* s to a destination t using geographic location infor-

mation, i. e., the coordinates of the nodes. It is assumed that the source node knows the coordinates of the destination node. A dedicated external location service is used for the source node to obtain this information [8]. The routing protocol consists of a sequence of communication steps. During each step, both the label of a unique transmitting node as well as the label of one of its neighbors who is expected to receive the transmitted message are specified by the routing protocol. Geometric routing is uniform in the context that all nodes execute the same protocol when deciding to which other node to forward a message.

Three classes of geometric routing algorithms are considered: *on-line geometric routing*, *off-line geometric routing* and *dynamic geometric routing*. In the context of all three classes, the focus is on routing the message from the source node to the destination using as small number of communication steps as possible. Note that the number of communication steps corresponds to the total number of transmissions. Thus by minimizing the number of communication steps, the number of transmissions is also minimized resulting in reduced power consumption. In what follows a list of combinatorial and algorithmic definitions commonly used in the context of geometric routing is given.

Planar Graph A graph $G = (V, E)$ is *planar* if nodes in V can be *embedded* into a 2-dimensional Euclidean Space $\mathcal{R}^2$, i. e., each node in V obtains a unique coordinates and an edge is drawn between every pair of nodes in E, in such way the resulting edges do not cross each other in $\mathcal{R}^2$.

Unit-Disk Graph (UDG) is defined to be a graph $G = (V, E)$ embedded into $\mathcal{R}^2$ where two nodes $u, v \in V$ are connected by an edge e if the Euclidean distance between u and v, denoted by $|u, v|$, is not greater than one.

λ-Precision/$\Omega(1)$ Model or Civilized Graph is defined to be a graph $G = (V, E)$ embedded into $\mathcal{R}^2$ where for any fixed $\lambda > 0$, two nodes $u, v \in V$ are of a distance at least λ apart.

Gabriel Graph (GG) is defined to be a graph $G = (V, E)$ embedded into $\mathcal{R}^2$ where for any $u, v \in V$ an edge $(u, v) \in E$ if u and v are the only nodes in V belonging to the circle with (u, v) as diameter.

Delaunay Triangulation Δ of a set of nodes V embedded into $\mathcal{R}^2$ is the geometric dual of the *Voronoi diagram* [9] of V, where any two nodes in V are linked by an edge in Δ if their corresponding cells in the Voronoi diagram are incident. A Delaunay triangulation Δ is *unit* if it contains edges of length at most one.

The Right Hand Principle is a rule used by graph traversal algorithms that primarily chooses the first edge to the right while moving towards the destination.

Heap-Like Structure Let $G = (V, E)$ be an undirected planar graph, s.t., each node in V contains some numerical value. A *heap-like structure* is a BFS tree T spanning all nodes in G, s.t., for every node v other than the root, the value stored at v is smaller than the value stored at v's parent.

Systems of clusters [2] Let $G = (V, E)$ be an undirected planar graph with $|V| = n$ and radius R. One can construct a system of families of clusters $F(0)$, $F(1), \ldots, F(\log R)$, s.t., (a) the diameter of each cluster in $F(i)$ is $O(2^i \log n)$, (b) every node belongs to at most $O(\log n)$ clusters, and (c) for any two nodes whose distance in G is $2^{i-1} < d \leq 2^i$, there exists at least one cluster in $F(i)$ that contains the two nodes.

Key Result and Applications

The key results on geometric routing rely on the following lemmas about Delaunay triangulation, planar graph and unit disk graph.

Lemma 1 ([9]) *The Delaunay triangulation Δ for a set of points V of cardinality n can be computed locally in time $O(n \log n)$.*

Lemma 2 ([4]) *Consider any $s, t \in V$. Assume x and y are two points such that s, x and y belong to a Delaunay triangulation Δ. And let α and β be the angles formed by segments $\overline{xs}$ and $\overline{st}$, and by segments $\overline{ts}$ and $\overline{sy}$ respectively. If $\alpha < \beta$, then $|xs| < |st|$. Otherwise $|ys| < |st|$.*

Lemma 3 *Let $G = (V, E)$ be a planar graph embedded into $\mathcal{R}^2$ and $s, t \in V$. Further, let x_i be the closest to t intersection point defined by some edge e_i belonging to some face F_i and the line segment $\overline{st}$. Similarly, let x_{i+1} be the closest to t intersection point defined by some edge belonging to face F_{i+1} and the line segment $\overline{st}$, where F_{i+1} is the face incident to F_i via edge e_i. Then $|x_i, t| > |x_{i+1}, t|$.*

Lemma 4 ([6]) *Let $G = (V, E)$ be a planar civilized graph embedded into $\mathcal{R}^2$. Any ellipse with major axis c covers at most $O(c^2)$ nodes and edges.*

Lemma 5 ([5]) *Let R be a convex region in $\mathcal{R}^2$ with area $A(R)$ and perimeter $P(R)$, and let $V \subset R$. If the unit disk graph of V has maximum degree k, the number of nodes in V is bounded by $|V| \leq 8(k+1)(A(R) + P(R) + \pi)/\pi$.*

Lemma 6 ([2]) *The number of transmissions required to construct a heap-like structure and the system of clusters for a planar graph G is bounded by $O(nD)$ and $O(n^2 D)$, respectively, where n is the number of nodes and D is the diameter of G.*

Applications

On-Line Geometric Routing

On-line geometric routing is based on very limited control information possessed by the routed message and the local information available at the network nodes. This results in natural scalability of geometric routing. It is also assumed that the network is static, i. e., the nodes do not move and no edges disappear nor (re)appear.

Compass Routing I (*CR-I*) [4] is a greedy procedure based on Delaunay triangulation and the observation from Lemma 2, where during each step the message is always routed to a neighboring node which is closer to the destination t. Unfortunately, the message may eventually end up in a local minimum (*dead end*) where all neighbors are further away from t. *CR-I* is very simple. Also computation of Delaunay triangulation is local and cheap, see Lemma 1. However, the algorithm does not guarantee successful delivery.

Compass Routing II (*CR-II*) [1,4] is the first geometric routing algorithm based on the right hand principle and the observation from Lemma 3 which guarantee successful delivery in any graph embedded into $\mathcal{R}^2$. The algorithm is also known as *Face Routing* since the routed message traverses along perimeters of faces closer and closer to the destination. In convex graph, the segment $\overline{st}$ intersects the perimeter of any face at most twice. Thus when the routed message hits the first edge e that intersects $\overline{st}$, it immediately changes the face to the other side of e. In consequence, every edge in each face is traversed at most twice. However, in general graph the routed message has to visit all edges incident to the face. This is to find the closest intersection point x_i to the destination t. In this case each edge can be visited even 4 times. However if after the traversal of all edges the routed message chooses the shorter path to x_i (rather than using the right hand principle), the amortized traversal cost of each edge is 3 [1]. The proof of correctness follows from Lemma 3.

Theorem 7 ([1]) *Compass Routing II guarantees successful delivery in planar graphs using $O(n)$ time where n is the number of nodes in the network.*

Adaptive Face Routing (*AFR*) [6] is an asymptotically optimal geometric routing in planar civilized graphs. The algorithm attempts to estimate the length c of the shortest path between s and t by $\widehat{c}$ (starting with $\widehat{c} = 2|\overline{st}|$ and doubling it in every consecutive round). In each round, the face traversal is restricted to the region formed by the ellipse with the major axis $\widehat{c}$ centered in $\overline{st}$. In AFR each edge is traversed at most 4 times, and the time complexity of *AFR* is $O(c^2)$, see Lemma 4. The corresponding lower bound is also provided in [6].

Theorem 8 ([6]) *The time complexity $O(c^2)$, where c is the length of the shortest path between s and t, is asymptotically optimal in civilized Unit Disk Graphs possessing Gabriel Graph properties.*

Geometric Ad-hoc Routing (*GOAFR*$^+$) [5] has provably good theoretical and practical performance. Due to Lemma 5, rather non-practical Ω (1) assumption can be dropped. *GOAFR*$^+$ combines greedy routing and face routing algorithms. The algorithm starts with the greedy routing *CR-I* and when the routed message enters a local minimum (*dead end*), it switches to *Face Routing*.

However, *GOAFR*$^+$ intends to return to greedy routing as early as possible via application of *early fallback* technique. The simulations show that *GOAFR*$^+$ outperforms *GOAFR* and *GOAFR*$_{FC}$ considered in [7] in the average case.

Theorem 9 ([2]) *GOAFR*$^+$ *has the optimal time complexity $O(c^2)$ in any Unit Disk Graphs possessing Gabriel Graph properties.*

Off-Line Geometric Routing

In off-line geometric routing, the routing stage is preceded by the preprocessing stage, when several data structures are constructed on the basis of the input graph G. This is to speed up the routing phase. The preprocessing is worthwhile if it is followed by further frequent queries.

Single-Source Queries [2] is a routing mechanism that allows to route messages from a distinguished source s to any other node t in the network in time $O(c)$, where c is the distance between s and t in G. The routing procedure is based on indirect addressing mechanism implemented in a heap-like structure that can be efficiently computed, see Lemma 6.

Multiple-Source-Queries [2] is an extension of the single-source querying mechanism that provides $O(c \log n)$-time routing between any pair of nodes located at distance c in G, where n is the number of nodes in G. The extension is based on the system of clusters that can be computed efficiently, see Lemma 6.

Theorem ([5]) *After preprocessing, single-source queries take time $O(c)$ and multiple-source queries take time $O(c \log n)$ in Unit Disk Graphs possessing Gabriel Graph properties.*

Dynamic Geometric Routing

Geometric Routing in Graphs with Dynamic Edges [3] applies to the model in which the nodes are fault-free and

stationary but the edges alternate their status between *active* and *inactive*. However, it is assumed that despite dynamic changes in the topology the network always remains connected. In this model *Timestamp-Traversal* routing algorithm combines the use of the global time and the starting time of the routing to traverse a spanning subgraph containing only stable links.

An alternative solution called *Tethered-Traversal* is based on the observation that (re)appearing edges potentially shorten the traversal paths, where the time/space complexity of the routing procedure is linear in the number of nodes n.

Open Problems

Very little is known about space efficient on-line routing in static *directed* graphs. Also the current bounds in dynamic geometric routing appear to be far from optimal.

Cross References

- ▶ Communication in Ad Hoc Mobile Networks Using Random Walks
- ▶ Minimum k-Connected Geometric Networks

Recommended Reading

1. Bose, P., Morin, P., Stojmenovic, I., Urrutia, J.: Routing with guaranteed delivery in ad hoc wireless networks. In: Proceedings of the Third International Workshop on Discrete Algorithm and Methods for Mobility, Seattle, Washington, Aug 1999, pp. 48–55
2. Gasieniec, L., Su, C., Wong, P.W.H., Xin, Q.: Routing via single-source and multiple-source queries in static sensor networks. J. Discret. Algorithm **5**(1), 1–11 (2007). A preliminary version of the paper appeared in IPDPS'2005
3. Guan, X.Y.: Face traversal routing on edge dynamic graphs. In: Proceedings of the Nineteenth International Parallel and Distributed Processing Symposium, Denver, Colorado, April 2005
4. Kranakis, E., Singh, H., Urrutia, J.: Compass routing on geometric networks. In: Proceedings of the Eleventh Canadian Conference on Computational Geometry, Vancover, BC, Canada, Aug 1999, pp. 51–54
5. Kuhn, F., Wattenhofer, R., Zhang, Y., Zollinger, A.: Geometric ad-hoc routing: Of theory and practice. In: Proceedings of the Twenty-Second ACM Symposium on the Principles of Distributed Computing, Boston, Massachusetts, July 2003, pp. 63–72
6. Kuhn, F., Wattenhofer, R., Zollinger, A.: Asymptotically optimal geometric mobile ad-hoc routing. In: Proceedings of the Sixth International Workshop on Discrete Algorithm and Methods for Mobility, Atlanta, Georgia, USA, Sept 2002, pp. 24–33
7. Kuhn, F., Wattenhofer, R., Zollinger, A.: Worst-case optimal and average-case efficient geometric ad-hoc routing. In: Proceedings of the Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing, Annapolis, Maryland, June 2003, pp. 267–278
8. Li, J., Jannotti, J., De Couto, D.S.J., Karger, D.R., Morris, R.: A scalable location service for geographic ad hoc routing. In Proceedings of the Sixth International Conference on Mobile Computing and Networking, Boston, Massachusetts, Aug 2000, pp. 120–130
9. Li, M., Lu, X.C., Peng, W.: Dynamic delaunay triangulation for wireless ad hoc network. In Proceedings of the Sixth International Workshop on Advanced Parallel Processing Technologies, Hong Kong, China, Oct 2005, pp. 382–389

Routing in Road Networks with Transit Nodes

2007; Bast, Funke, Sanders, Schultes

Dominik Schultes
Institute for Computer Science,
University of Karlsruhe, Karlsruhe, Germany

Keywords and Synonyms

Shortest paths

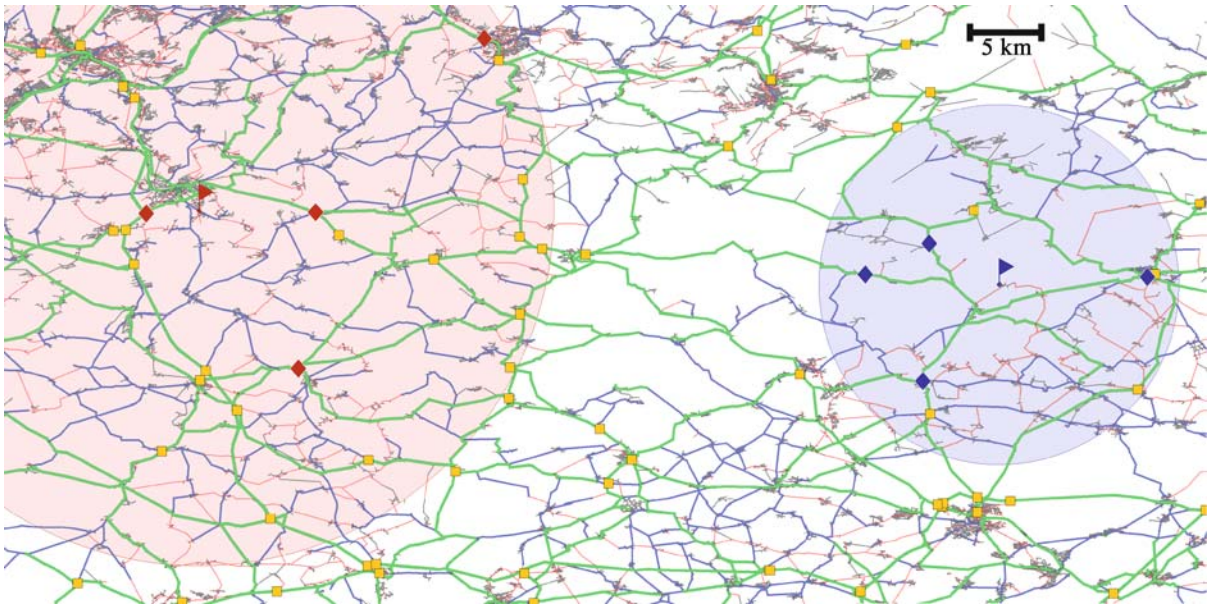
Problem Definition

For a given directed graph $G = (V, E)$ with non-negative edge weights, the problem is to compute a shortest path in G from a source node s to a target node t for given s and t. Under the assumption that G does not change and that a lot of source-target queries have to be answered, it pays to invest some time for a preprocessing step that allows for very fast queries. As output, either a full description of the shortest path or only its length $d(s, t)$ is expected—depending on the application.

Dijkstra's classical algorithm for this problem [4] iteratively visits all nodes in the order of their distance from the source until the target is reached. When dealing with very large graphs, this general algorithm gets too slow for many applications so that more specific techniques are needed that exploit special properties of the particular graph. One practically very relevant case is routing in road networks where junctions are represented by nodes and road segments by edges whose weight is determined by some weighting of, for example, expected travel time, distance, and fuel consumption. Road networks are typically sparse (i. e., $|E| = O(|V|)$), almost planar (i. e., there are only a few overpasses), and hierarchical (i. e., more or less 'important' roads can be distinguished). An overview on various speedup techniques for this specific problem is given in [7].

Key Results

Transit-node routing [2,3] is based on a simple observation intuitively used by humans: When you start from a source node s and drive to somewhere 'far away', you will leave



Routing in Road Networks with Transit Nodes, Figure 1
Finding the optimal travel time between two points (*flags*) somewhere between Saarbrücken and Karlsruhe amounts to retrieving the 2 × 4 *access nodes* (*diamonds*), performing 16 table lookups between all pairs of access nodes, and checking that the two disks defining the *locality filter* do not overlap. *Transit nodes* that do not belong to the access node sets of the selected source and target nodes are drawn as small squares. The figure draws the levels of the highway hierarchy using colors gray, red, blue, and green for levels 0–1, 2, 3, and 4, respectively

your current location via one of only a few 'important' traffic junctions, called (forward) *access nodes* $\overrightarrow{A}(s)$. An analogous argument applies to the target t, i. e., the target is reached from one of only a few backward access nodes $\overleftarrow{A}(t)$. Moreover, the union of all forward and backward access nodes of all nodes, called *transit-node set* $\mathcal{T}$, is rather small. The two observations imply that for each node the distances to/from its forward/backward access nodes and for each transit-node pair (u, v), the distance between u and v can be stored. For given source and target nodes s and t, the length of the shortest path that passes at least one transit node is given by

$$d_{\mathcal{T}}(s,t) = \min\{d(s,u) + d(u,v) + d(v,t) \mid u \in \overrightarrow{A}(s), v \in \overleftarrow{A}(t)\}\,.$$

Note that all involved distances $d(s, u)$, $d(u, v)$, and $d(v, t)$ can be directly looked up in the precomputed data structures. As a final ingredient, a *locality filter* $L : V \times V \to \{\mathsf{true}, \mathsf{false}\}$ is needed that decides whether given nodes s and t are too close to travel via a transit node. L has to fulfill the property that $\neg L(s, t)$ implies that $d(s, t) = d_{\mathcal{T}}(s, t)$. Note that in general the converse need not hold since this might hinder an efficient realization of the locality filter. Thus, *false positives*, i. e., "$L(s, t) \wedge d(s, t) = d_{\mathcal{T}}(s, t)$", may occur.

The following algorithm can be used to compute $d(s, t)$:

If $\neg L(s, t)$, then compute and return $d_{\mathcal{T}}(s, t)$;
else, use any other routing algorithm.

Figure 1 gives an example. Knowing the length of the shortest path, a complete description of it can be efficiently derived using iterative table lookups and precomputed representations of paths between transit nodes. Provided that the above observations hold and that the percentage of false positives is low, the above algorithm is very efficient since a large fraction of all queries can be handled in line 1, $d_{\mathcal{T}}(s, t)$ can be computed using only a few table lookups, and source and target of the remaining queries in line 2 are quite close. Indeed, the remaining queries can be further accelerated by introducing a secondary layer of transit-node routing, based on a set of *secondary transit nodes* $\mathcal{T}_2 \supset \mathcal{T}$. Here, it is not necessary to compute and store a complete $\mathcal{T}_2 \times \mathcal{T}_2$ distance table, but it is sufficient to store only distances $\{d(u, v) \mid u, v \in \mathcal{T}_2 \wedge d(u, v) \neq d_{\mathcal{T}}(s, t)\}$, i. e., distances that cannot be obtained using the primary layer. Analogously, further layers can be added.

Routing in Road Networks with Transit Nodes, Table 1

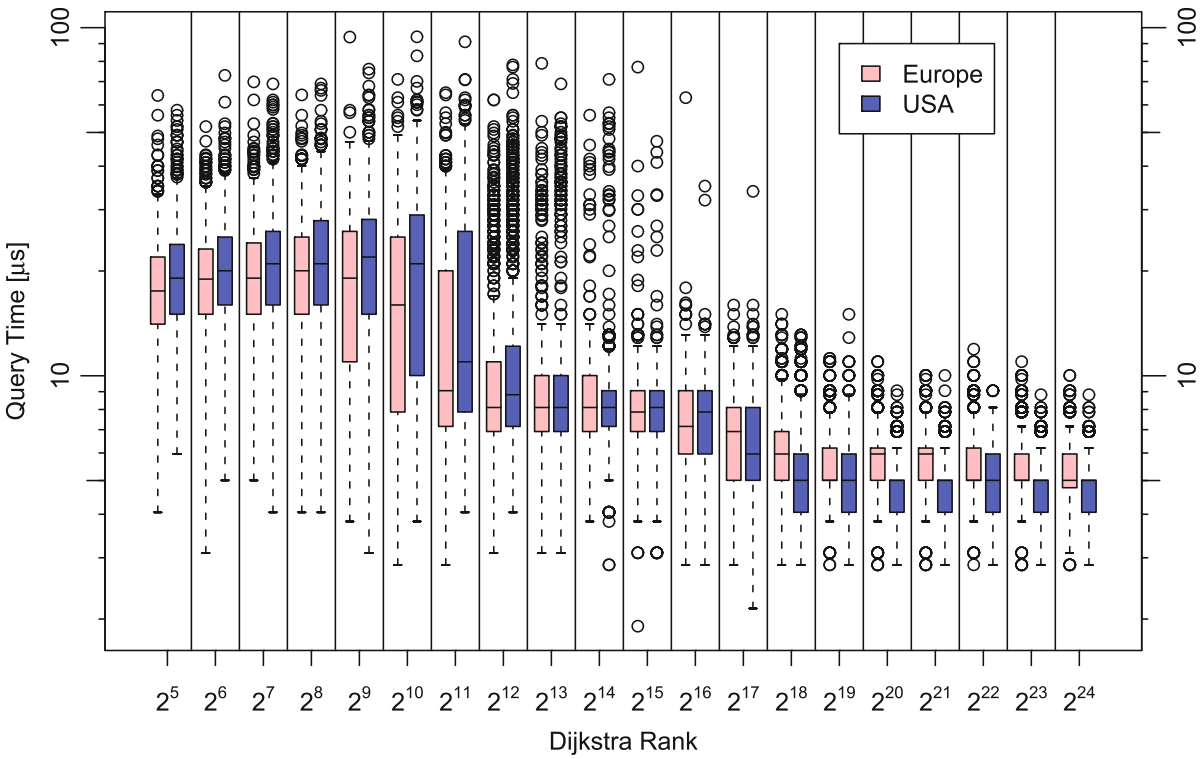
Statistics on preprocessing. The size of transit-node sets, the number of entries in distance tables, and the average number of access nodes to the respective layer are given; furthermore, the space overhead and the preprocessing time

	layer 1		layer 2			layer 3			
	$\lvert\mathcal{T}\rvert$	$\lvert A\rvert$ avg.	$\lvert\mathcal{T}_2\rvert$	$\lvert\text{table}_2\rvert$ [$\times 10^6$]	$\lvert A_2\rvert$ avg.	$\lvert\mathcal{T}_3\rvert$	$\lvert\text{table}_3\rvert$ [$\times 10^6$]	space [B/node]	time [h]
Europe	11 293	9.9	323 356	130	4.1	2 954 721	119	251	2:44
USA	10 674	5.7	485 410	204	4.2	3 855 407	173	244	3:25

Routing in Road Networks with Transit Nodes, Table 2

Performance of transit-node routing with respect to 10 000 000 random queries. The column for layer *i* specifies which fraction of the queries is correctly answered using only information available at layers $\leq i$. Each box spreads from the lower to the upper quartile and contains the median, the whiskers extend to the minimum and maximum value omitting outliers, which are plotted individually

	#nodes	#edges	layer 1	layer 2	layer 3	query
Europe	18 029 721	42 199 587	99.74%	99.9984%	99.99981%	5.6 μs
USA	24 278 285	58 213 192	99.89%	99.9986%	99.99986%	4.9 μs



Routing in Road Networks with Transit Nodes, Figure 2

Query time distribution as a function of Dijkstra rank–the number of iterations Dijkstra's algorithm would need to solve this instance. The distributions are represented as box-and-whisker plots: each box spreads from the lower to the upper quartile and contains the median, the whiskers extended to the minimum and maximum value omitting, which are plotted individually

There are two different implementations: one is based on a simple geometric grid and one on *highway hierarchies*, the fastest previous approach [5,6]. A highway hierarchy consists of a sequence of levels (Fig. 1), where level $i+1$ is constructed from level i by bypassing low-degree nodes and removing edges that never appear far away from the source or target of a shortest path. Interestingly, these levels are geometrically decreasing in size and otherwise

similar to each other. The highest level contains the most 'important' nodes and becomes the primary transit-node set. The nodes of lower levels are used to form the transit-node sets of subordinated layers.

Applications

Apart from the most obvious applications in car navigation systems and server-based route planning systems, transit-node routing can be applied to several other fields, for instance to massive traffic simulations and to various optimization problems in logistics.

Open Problems

It is an open question whether one can find better transit-node sets or a better locality filter so that the performance can be further improved. It is also not clear if transit-node routing can be successfully applied to other graph types than road networks. In this context, it would be desirable to derive some theoretical guarantees that apply to any graph that fulfills certain properties. For some practical applications, a *dynamic* version of transit-node routing would be required in order to deal with time-dependent networks or unexpected edge weight changes caused, for example, by traffic jams. The latter scenario can be handled by a related approach [8], which is, however, considerably slower than transit-node routing.

Experimental Results

Experiments were performed on road networks of Western Europe and the USA using a cost function that solely takes expected travel time into account. The results exhibit various tradeoffs between average query time (5 µs to 63 µs for the USA), preprocessing time (59 min to 1200 min), and storage overhead (21 bytes/node to 244 bytes/node). For the variant that uses three layers and is tuned for best query times, Tables 1 and 2 show statistics on the preprocessing and the query performance, respectively. The average query times of about 5 µs are six orders of magnitude faster than Dijkstra's algorithm. In addition, Fig. 2 gives for each rank r on the x-axis a distribution for 1 000 queries with random starting point s and the target node t for which Dijkstra's algorithm would need r iterations to find it. The three layers of transit-node routing with small transition zones in between can be recognized: for large ranks, it is sufficient to access only the primary layer yielding query times of about 5 µs, for smaller ranks, additional layers have to be accessed resulting in median query times of up to 20 µs.

Data Sets

The European road network has been provided by the company PTV AG, the US network has been obtained from the TIGER/Line Files [9]. Both graphs have also been used in the 9th DIMACS Implementation Challenge on Shortest Paths [1].

URL to Code

The source code might be published at some point in the future at http://algo2.iti.uka.de/schultes/hwy/.

Cross References

- ▶ All Pairs Shortest Paths in Sparse Graphs
- ▶ All Pairs Shortest Paths via Matrix Multiplication
- ▶ Decremental All-Pairs Shortest Paths
- ▶ Engineering Algorithms for Large Network Applications
- ▶ Fully Dynamic All Pairs Shortest Paths
- ▶ Geographic Routing
- ▶ Implementation Challenge for Shortest Paths
- ▶ Shortest Paths Approaches for Timetable Information
- ▶ Shortest Paths in Planar Graphs with Negative Weight Edges
- ▶ Single-Source Shortest Paths

Recommended Reading

1. 9th DIMACS Implementation Challenge: Shortest Paths. http://www.dis.uniroma1.it/~challenge9/ (2006)
2. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In transit to constant time shortest-path queries in road networks. In: Workshop on Algorithm Engineering and Experiments, 2007, pp. 46–59
3. Bast, H., Funke, S., Sanders, P., Schultes, D.: Fast routing in road networks with transit nodes. Science **316**(5824), 566 (2007)
4. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1** 269–271 (1959)
5. Sanders, P., Schultes, D.: Highway hierarchies hasten exact shortest path queries. In: 13th European Symposium on Algorithms. LNCS, vol. 3669, pp. 568–579. Springer, Berlin (2005)
6. Sanders, P., Schultes, D.: Engineering highway hierarchies. In: 14th European Symposium on Algorithms. LNCS, vol. 4168, pp. 804–816. Springer, Berlin (2006)
7. Sanders, P., Schultes, D.: Engineering fast route planning algorithms. In: 6th Workshop on Experimental Algorithms. LNCS, vol. 4525, pp. 23–36. Springer, Berlin (2007)
8. Schultes, D., Sanders, P.: Dynamic highway-node routing. In: 6th Workshop on Experimental Algorithms. LNCS, vol. 4525, pp. 66–79. Springer, Berlin (2007)
9. U.S. Census Bureau, Washington, DC: UA Census 2000 TIGER/Line Files. http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html (2002)

R-Trees

2004; Arge, de Berg, Haverkort, Yi

Ke Yi
Hong Kong University of Science and Technology, Hong Kong, China

Keywords and Synonyms

R-Trees; Spatial databases; External memory data structures

Problem Definition

Problem Statement and the I/O Model

Let S be a set of N axis-parallel hypercubes in $\mathbb{R}^d$. A very basic operation in a spatial database is to answer *window queries* on the set S. A window query Q is also an axis-parallel hypercube in $\mathbb{R}^d$ that asks us to return all hypercubes in S that intersect Q. Since the set S is typically huge in a large spatial database, the goal is to design a *disk-based*, or *external memory* data structure (often called an *index* in the database literature) such that these window queries can be answered efficiently. In addition, given S, the data structure should be constructed efficiently, and should be able to support insertions and deletions of objects.

When external memory data structures are concerned, the standard *external memory model* [2], a.k.a. the *I/O model*, is often used as the model of computation. In this model, the machine consists of an infinite-size external memory (disk) and a main memory of size M. A block of B consecutive elements can be transferred between main memory and disk in one *I/O operation* (or simply *I/O*). An external memory data structure is a structure that is stored on disk in blocks, but computation can only occur on elements in main memory, so any operation (e. g. query, update, and construction) on the data structure must be performed using a number I/Os, which is the measure for the complexity of the operation.

R-Trees

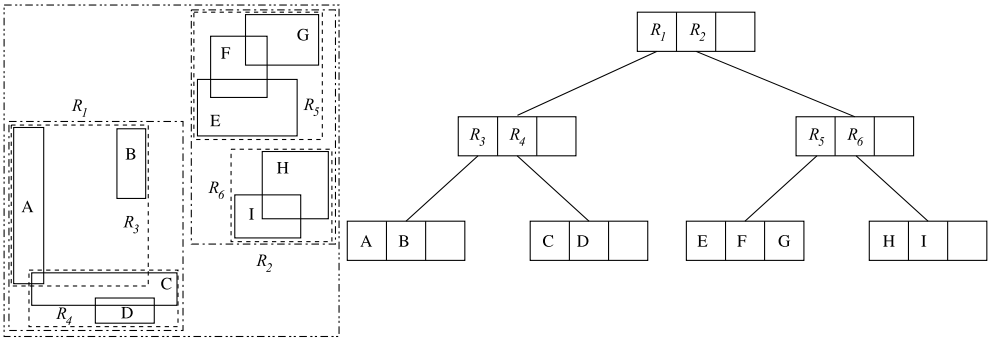
The *R-tree*, first proposed by Guttman [9], is a multi-way tree $\mathcal{T}$, very similar to a *B-tree*, that is used to store the set S such that a window query can be answered efficiently. Each node of $\mathcal{T}$ fits in one disk block. The hypercubes of S are stored only in the leaves of $\mathcal{T}$. All leaves of $\mathcal{T}$ are on the same level, and each stores $\Theta(B)$ hypercubes from S; while each internal node, except the root, has a fan-out of $\Theta(B)$. The root of $\mathcal{T}$ may have a fan-out as small as 2. For any node $u \in \mathcal{T}$, let $R(u)$ be the smallest axis-parallel hypercube, called the *minimal bounding box*, that encloses all the hypercubes stored below u. At each internal node $v \in \mathcal{T}$, whose children are denoted $v_1,\ldots,v_k$, the bounding box $R(v_i)$ is stored along with the pointer to v_i for $i = 1,\ldots,k$. Note that these bounding boxes may overlap. Please see Fig. 1 for an example of an R-tree in two dimensions.

For a window query Q, the query answering process starts from the root of $\mathcal{T}$ and visits all nodes u for which $R(u)$ intersects Q. When reaching a leaf v, it checks each hypercube stored at v to decide if it should be reported. The correctness of the algorithm is obvious, and the efficiency (the number of I/Os) is determined by the number of nodes visited.

Any R-tree occupies a linear number $O(N/B)$ disk blocks, but different R-trees might have different query, update, and construction costs. When analyzing the query complexity of window queries, the output size T is also used, in addition to N, M, and B.

Key Results

Although the structure of an R-tree is restricted, there is much freedom in grouping the hypercubes into leaves and grouping subtrees into bigger subtrees. Different grouping strategies result in different variants of R-trees. Most of the existing R-trees use various heuristics to group together hypercubes that are "close" spatially, so that a window query will not visit too many unnecessary nodes. Generally speaking, there are two ways to build an R-tree: repeated insertion and bulk-loading. The former type of algorithms include the original R-tree [9], the R$^+$-tree [15], the R*-tree [6], etc. These algorithms use $O(\log_B N)$ I/Os to insert an object and hence $O(N \log_B N)$ I/Os to build the R-tree on S, which is not scalable for large N. When the set S is known in advance, it is much more efficient to bulk-load the entire R-tree at once. Many bulk-loading algorithms have been proposed, e. g. [7,8,10,11,13]. Most of these algorithms build the R-tree with $O(N/B \log_{M/B} N/B)$ I/Os (the number of I/Os needed to sort N elements), and they typically result in better R-trees than those obtained by repeated insertion. During the past decades, there have been a large number of works devoted to R-trees from the database community, and the list here is by no means complete. The reader is referred to the book by Manolopoulos et al. [14] for an excellent survey on this subject in the database literature. However, no R-tree variant mentioned above has a guarantee on the query complexity; in fact, Arge et al. [3] constructed an example showing that some of the most popular R-trees may have to visit all the nodes without reporting a single result.



R-Trees, Figure 1
An R-tree example in two dimensions

From the theoretical perspective, the following are the two main results concerning the worst-case query complexity of R-trees.

Theorem 1 ([1,12]) *There is a set of N points in $\mathbb{R}^d$, such that for any R-tree $\mathcal{T}$ built on these points, there exists an empty window query for which the query algorithm has to visit $\Omega((N/B)^{1-1/d})$ nodes of $\mathcal{T}$.*

The *priority R-tree*, proposed by Arge et al. [3], matches the above lower bound.

Theorem 2 ([3]) *For any set S of N axis-parallel hypercubes in $\mathbb{R}^d$, the priority R-tree answers a window query with $O((N/B)^{1-1/d} + T/B)$ I/Os. It can be constructed with $O(N/B \log_{M/B} N/B)$ I/Os.*

It is also reported that the priority R-tree performs well in practice, too [3]. However, it is not known how to update it efficiently while preserving the worst-case bound. The logarithmic method was used to support insertions and deletions [3] but the resulted structure is no longer an R-tree.

Note that the lower bound in Theorem 1 only holds for R-trees. If the data structure is not restricted to R-trees, better query bounds can be obtained for the window-query problem; see e. g. [4].

Applications

R-trees have been used widely in practice due to its simplicity, the ability to store spatial objects of various shapes, and to answer various queries. The areas of applications span from geographical information systems (GIS), computer-aided design, computer vision, and robotics. When the objects are not axis-parallel hypercubes, they are often approximated by their minimal bounding boxes, and the R-tree is then built on these bounding boxes. To answer a window query, first the R-tree is used to locate all the intersecting bounding boxes, followed by a filtering step that checks the objects exactly. The R-tree can also be used to support other kinds of queries, for example aggregation queries, nearest-neighbors, etc. In aggregation queries, each object o in S is associated with a weight $w(o) \in \mathbb{R}$, and the goal is to compute $\sum w(o)$ where the sum is taken over all objects that intersect the query range Q. The query algorithm is same as before, except that in addition it keeps running sum while traversing the R-tree, and may skip an entire subtree rooted at some u if $R(u)$ is completely contained in Q. To find the nearest neighbor of a query point q, a priority queue is maintained, which stores all the nodes u that might contain an object that is closer to the current nearest neighbor found so far. The priority of u in the queue is the distance between q and $R(u)$. The search terminates when the current nearest neighbor is closer than the top element in the priority queue. However, no worst-case guarantees are known for R-trees answering these other types of queries, although they tend to perform well in practice.

Open Problems

Several interesting problems remain open with respect to R-trees. Some of them are listed here.

- Is it possible to design an R-tree with the optimal query bound $O((N/B)^{1-1/d} + T/B)$ that can also be efficiently updated? Or prove a lower bound on the update cost for such an R-tree.
- Is there an R-tree that supports aggregation queries for axis-parallel hypercubes in $O((N/B)^{1-1/d})$ I/Os? This would be optimal because the lower bound of Theorem 1 also holds for aggregation queries on R-trees. Note that, however, no sub-linear worst-case bound exists for nearest-neighbor queries, since it is not difficult to design a worst-case example for which the distance

between the query point q and any bounding box is smaller than the distance between q and its true nearest neighbor.

- When the window query Q shrinks to a point, that is, the query asks for all hypercubes in S that contain the query point, the problem is often referred to as *stabbing queries* or *point enclosure queries*. The lower bound of Theorem 1 does not hold for this special case; while a lower bound of $\Omega(\log_2 N + T/B)$ was proved in [5], which holds in the strong *indexability* model. It is intriguing to find out the true complexity for stabbing queries using R-trees, which is between $\Omega(\log_2 N + T/B)$ and $O((N/B)^{1-1/d} + T/B)$.

Experimental Results

Nearly all studies on R-trees include experimental evaluations, mostly in two dimensions. Reportedly the Hilbert R-tree [10,11] has been shown to have good query performance while being easy to construct. The R*-tree's insertion algorithm [6] has often been used for updating the R-tree. Please refer to the book by Manolopoulos et al. [14] for more discussions on the practical performance of R-trees.

Data Sets

Besides some synthetic data sets, the TIGER/Line data (http://www.census.gov/geo/www/tiger/) from the US Census Bureau has been frequently used as real-world data to test R-trees. The R-tree portal (http://www.rtreeportal.org/) also contains many interesting data sets.

URL to Code

Code for many R-tree variants is available at the R-tree portal (http://www.rtreeportal.org/). The code for the priority R-tree is available at http://www.cse.ust.hk/~yike/prtree/.

Cross References

- ► B-trees
- ► External Sorting and Permuting
- ► I/O-model

Recommended Reading

1. Agarwal, P.K., de Berg, M., Gudmundsson, J., Hammar, M., Haverkort, H.J.: Box-trees and R-trees with near-optimal query time. Discret. Comput. Geom. **28**, 291–312 (2002)
2. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. In: Communications of the ACM, vol. 31, pp. 1116–1127 (1988)
3. Arge, L., de Berg, M., Haverkort, H.J., Yi, K.: The priority R-tree: A practically efficient and worst-case optimal R-tree. In: Proc. SIGMOD International Conference on Management of Data, 2004, pp. 347–358
4. Arge, L., Samoladas, V., Vitter, J.S.: On two-dimensional indexability and optimal range search indexing. In: Proc. ACM Symposium on Principles of Database Systems, 1999, pp. 346–357
5. Arge, L., Samoladas, V., Yi, K.: Optimal external memory planar point enclosure. In: Proc. European Symposium on Algorithms, 2004
6. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-tree: An efficient and robust access method for points and rectangles. In: Proc. SIGMOD International Conference on Management of Data, 1990, pp. 322–331
7. DeWitt, D.J., Kabra, N., Luo, J., Patel, J.M., Yu, J.-B.: Client-server paradise. In: Proc. International Conference on Very Large Databases, 1994, pp. 558–569
8. García, Y.J., López, M.A., Leutenegger, S.T.: A greedy algorithm for bulk loading R-trees. In: Proc. 6th ACM Symposium on Advances in GIS, 1998, pp. 163–164
9. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proc. SIGMOD International Conference on Management of Data, 1984, pp. 47–57
10. Kamel, I., Faloutsos, C.: On packing R-trees. In: Proc. International Conference on Information and Knowledge Management, 1993, pp. 490–499
11. Kamel, I., Faloutsos, C.: Hilbert R-tree: An improved R-tree using fractals. In: Proc. International Conference on Very Large Databases, 1994, pp. 500–509
12. Kanth, K.V.R., Singh, A.K.: Optimal dynamic range searching in non-replicating index structures. In: Proc. International Conference on Database Theory. LNCS, vol. 1540, pp. 257–276 (1999)
13. Leutenegger, S.T., Lopez, M.A., Edington, J.: STR: A simple and efficient algorithm for R-tree packing. In: Proc. 13th IEEE International Conference on Data Engineering, 1997, pp. 497–506
14. Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A.N., Theodoridis, Y.: R-trees: Theory and Applications. Springer, London (2005)
15. Sellis, T., Roussopoulos, N., Faloutsos, C.: The R^+-tree: A dynamic index for multi-dimensional objects. In: Proc. International Conference on Very Large Databases, 1987, pp. 507–518

Runs

► Squares and Repetitions