Implementation Challenge for Shortest Paths

2006; Demetrescu, Goldberg, Johnson

Camil Demetrescu[1], Andrew V. Goldberg[2], David S. Johnson[3]

[1] Department of Information and Computer Systems, University of Roma, Rome, Italy
[2] Microsoft Research – Silicon Valley, Mountain View, CA, USA
[3] Algorithms and Optimization Research Dept., AT&T Labs, Florham Park, NJ, USA

Keywords and Synonyms

Test sets and experimental evaluation of computer programs for solving shortest path problems; DIMACS

Problem Definition

DIMACS Implementation Challenges (http://dimacs.rutgers.edu/Challenges/) are scientific events devoted to assessing the practical performance of algorithms in experimental settings, fostering effective technology transfer and establishing common benchmarks for fundamental computing problems. They are organized by DIMACS, the Center for Discrete Mathematics and Theoretical Computer Science. One of the main goals of DIMACS Implementation Challenges is to address questions of determining realistic algorithm performance where worst case analysis is overly pessimistic and probabilistic models are too unrealistic: experimentation can provide guides to realistic algorithm performance where analysis fails. Experimentation also brings algorithmic questions closer to the original problems that motivated theoretical work. It also tests many assumptions about implementation methods and data structures. It provides an opportunity to develop and test problem instances, instance generators, and other methods of testing and comparing performance of algorithms. And it is a step in technology transfer by providing leading edge implementations of algorithms for others to adapt.

The first Challenge was held in 1990–1991 and was devoted to *Network flows and Matching*. Other addressed problems included: *Maximum Clique, Graph Coloring, and Satisfiability* (1992–1993), *Parallel Algorithms for Combinatorial Problems* (1993–1994), *Fragment Assembly and Genome Rearrangements* (1994–1995), *Priority Queues, Dictionaries, and Multi-Dimensional Point Sets* (1995–1996), *Near Neighbor Searches* (1998–1999), *Semidefinite and Related Optimization Problems* (1999–2000), and *The Traveling Salesman Problem* (2000–2001).

This entry addresses the goals and the results of the *9th DIMACS Implementation Challenge*, held in 2005–2006 and focused on *Shortest Path* problems.

The 9th DIMACS Implementation Challenge: The Shortest Path Problem

Shortest path problems are among the most fundamental combinatorial optimization problems with many applications, both direct and as subroutines in other combinatorial optimization algorithms. Algorithms for these problems have been studied since the 1950's and still remain an active area of research.

One goal of this Challenge was to create a reproducible picture of the state of the art in the area of shortest path algorithms, identifying a standard set of benchmark instances and generators, as well as benchmark implementations of well-known shortest path algorithms. Another goal was to enable current researchers to compare their codes with each other, in hopes of identifying the more effective of the recent algorithmic innovations that have been proposed.

Challenge participants studied the following variants of the shortest paths problem:

- *Point to point shortest paths* [4,5,6,9,10,11,14]: the problem consists of answering multiple online queries about the shortest paths between pairs of vertices

and/or their lengths. The most efficient solutions for this problem preprocess the graph to create a data structure that facilitates answering queries quickly.

- *External-memory shortest paths* [2]: the problem consists of finding shortest paths in a graph whose size is too large to fit in internal memory. The problem actually addressed in the Challenge was single-source shortest paths in undirected graphs with unit edge weights.
- *Parallel shortest paths* [8,12]: the problem consists of computing shortest paths using multiple processors, with the goal of achieving good speedups over traditional sequential implementations. The problem actually addressed in the Challenge was single-source shortest paths.
- *K-shortest paths* [13,15]: the problem consists of ranking paths between a pair of vertices by non decreasing order of their length.
- *Regular-language constrained shortest paths:* [3] the problem consists of a generalization of shortest path problems where paths must satisfy certain constraints specified by a regular language. The problems studied in the context of the Challenge were single-source and point-to-point shortest paths, with applications ranging from transportation science to databases.

The Challenge culminated in a Workshop held at the DIMACS Center at Rutgers University, Piscataway, New Jersey on November 13–14, 2006. Papers presented at the conference are available at the URL: http://www.dis.uniroma1.it/~challenge9/papers.shtml. Selected contributions are expected to appear in a book published by the American Mathematical Society in the DIMACS Book Series.

Key Results

The main results of the 9th DIMACS Implementation Challenge include:

- Definition of common file formats for several variants of the shortest path problem, both static and dynamic. These include an extension of the famous DIMACS graph file format used by several algorithmic software libraries. Formats are described at the URL: http://www.dis.uniroma1.it/~challenge9/formats.shtml.
- Definition of a common set of core input instances for evaluating shortest path algorithms.
- Definition of benchmark codes for shortest path problems.
- Experimental evaluation of state-of-the-art implementations of shortest path codes on the core input families.
- A discussion of directions for further research in the area of shortest paths, identifying problems critical in real-world applications for which efficient solutions still remain unknown.

The chief information venue about the 9th DIMACS Implementation Challenge is the website http://www.dis.uniroma1.it/~challenge9.

Applications

Shortest path problems arise naturally in a remarkable number of applications. A limited list includes transportation planning, network optimization, packet routing, image segmentation, speech recognition, document formatting, robotics, compilers, traffic information systems, and dataflow analysis. It also appears as a subproblem of several other combinatorial optimization problems such as network flows. A comprehensive discussion of applications of shortest path problems appears in [1].

Open Problems

There are several open questions related to shortest path problems, both theoretical and practical. One of the most prominent discussed at the 9th DIMACS Challenge Workshop is modeling traffic fluctuations in point-to-point shortest paths. The current fastest implementations preprocess the input graph to answer point-to-point queries efficiently, and this operation may take hours on graphs arising in large-scale road map navigation systems. A change in the traffic conditions may require rescanning the whole graph several times. Currently, no efficient technique is known for updating the preprocessing information without rebuilding it from scratch. This would have a major impact on the performance of routing software.

Data Sets

The collection of benchmark inputs of the 9th DIMACS Implementation Challenge includes both synthetic and real-world data. All graphs are strongly connected. Synthetic graphs include random graphs, grids, graphs embedded on a torus, and graphs with small-world properties. Real-world inputs consist of graphs representing the road networks of Europe and USA. Europe graphs are provided by courtesy of the PTV company, Karlsruhe, Germany, subject to signing a (no-cost) license agreement. They include the road networks of 17 European countries: AUT, BEL, CHE, CZE, DEU, DNK, ESP, FIN, FRA, GBR, IRL, ITA, LUX, NDL, NOR, PRT, SWE, with a total of about 19 million nodes and 23 million edges. USA graphs are derived from the *UA Census 2000 TIGER/Line*

Implementation Challenge for Shortest Paths, Table 1
USA Road Networks derived from the TIGER/Line collection

NAME	DESCRIPTION	NODES	ARCS	BOUNDING BOX LATITUDE (N)	BOUNDING BOX LONGITUDE (W)
USA	Full USA	23 947 347	58 333 344	–	–
CTR	Central USA	14 081 816	34 292 496	[25.0; 50.0]	[79.0; 100.0]
W	Western USA	6 262 104	15 248 146	[27.0; 50.0]	[100.0; 130.0]
E	Eastern USA	3 598 623	8 778 114	[24.0; 50.0]	[-∞; 79.0]
LKS	Great Lakes	2 758 119	6 885 658	[41.0; 50.0]	[74.0; 93.0]
CAL	California and Nevada	1 890 815	4 657 742	[32.5; 42.0]	[114.0; 125.0]
NE	Northeast USA	1 524 453	3 897 636	[39.5, 43.0]	[-∞; 76.0]
NW	Northwest USA	1 207 945	2 840 208	[42.0; 50.0]	[116.0; 126.0]
FLA	Florida	1 070 376	2 712 798	[24.0; 31.0]	[79; 87.5]
COL	Colorado	435 666	1 057 066	[37.0; 41.0]	[102.0; 109.0]
BAY	Bay Area	321 270	800 172	[37.0; 39.0]	[121; 123]
NY	New York City	264 346	733 846	[40.3; 41.3]	[73.5; 74.5]

Implementation Challenge for Shortest Paths, Table 2
Results of the Challenge competition on the USA graph (23.9 million nodes and 58.3 million arcs) with unit arc lengths. The benchmark ratio is the average query time divided by the time required to answer a query using the Challenge Dijkstra benchmark code on the same platform. Query times and node scans are average values per query over 1000 random queries

	PREPROCESSING		QUERY		
CODE	Time (minutes)	Space (MB)	Node scans	Time (ms)	Benchmark ratio
HH-based transit [14]	104	3664	n.a.	0.019	$4.78 \cdot 10^{-6}$
TRANSIT [4]	720	n.a.	n.a.	0.052	$10.77 \cdot 10^{-6}$
HH Star [6]	32	2662	1082	1.14	$287.32 \cdot 10^{-6}$
REAL(16,1) [9]	107	2435	823	1.42	$296.30 \cdot 10^{-6}$
HH with DistTab [6]	29	2101	1671	1.61	$405.77 \cdot 10^{-6}$
RE [9]	88	861	3065	2.78	$580.08 \cdot 10^{-6}$

Files produced by the Geography Division of the US Census Bureau, Washington, DC. The TIGER/Line collection is available at: http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html. The Challenge USA core family contains a graph representing the full USA road system with about 24 million nodes and 58 million edges, plus 11 subgraphs obtained by cutting it along different bounding boxes as shown in Table 1. Graphs in the collection include also node coordinates and are given in DIMACS format.

The benchmark input package also features query generators for the single-source and point-to-point shortest path problems. For the single-source version, sources are randomly chosen. For the point-to-point problem, both random and local queries are considered. Local queries of the form (s, t) are generated by randomly picking t among the nodes with rank in $[2^i, 2^{i+1})$ in the ordering in which nodes are scanned by Dijkstra's algorithm with source s, for any parameter i. Clearly, the smaller i is, the closer nodes s and t are in the graph. Local queries are important to test how the algorithms' performance is affected by the distance between query endpoints.

The core input families of the 9th DIMACS Implementation Challenge are available at the URL: http://www.dis.uniroma1.it/~challenge9/download.shtml.

Experimental Results

One of the main goals of the Challenge was to compare different techniques and algorithmic approaches. The most popular topic was the point-to-point shortest path problem, studied by six research groups in the context of the Challenge. For this problem, participants were additionally invited to join a competition aimed at assessing the performance and the robustness of different implementations. The competition consisted of preprocessing a version of the full USA graph of Table 1 with unit edge lengths and answering a sequence of 1,000 random distance queries. The details were announced on the first day of the workshop and the results were due on the second day. To compare experimental results by different participants on different platforms, each participant ran a Dijkstra benchmark code [7] on the USA graph to do machine

calibration. The final ranking was made by considering each query time divided by the time required by the benchmark code on the same platform (benchmark ratio). Other performance measures taken into account were space usage and the average number of nodes scanned by query operations.

Six point-to-point implementations were run successfully on the USA graph defined for the competition. Among them, the fastest query time was achieved by the *HH-based transit* code [14]. Results are reported in Table 2. Codes *RE* and *REAL*(16, 1) [9] were not eligible for the competition, but used by the organizers as a proof that the problem is feasible. Some other codes were not able to deal with the size of the full USA graph, or incurred run-time errors.

Experimental results for other variants of the shortest paths problem are described in the papers presented at the Challenge Workshop.

URL to Code

Generators of problem families and benchmark solvers for shortest paths problems are available at the URL: http://www.dis.uniroma1.it/~challenge9/download.shtml.

Cross References

- ► Engineering Algorithms for Large Network Applications
- ► Experimental Methods for Algorithm Analysis
- ► High Performance Algorithm Engineering for Large-scale Problems
- ► Implementation Challenge for TSP Heuristics
- ► LEDA: a Library of Efficient Algorithms

Recommended Reading

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows: Theory, Algorithms and Applications. Prentice Hall, Englewood Cliffs, NJ (1993)
2. Ajwani, D., Dementiev, U., Meyer, R., Osipov, V.: Breadth first search on massive graphs. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
3. Barrett, C., Bissett, K., Holzer, M., Konjevod, G., Marathe, M., Wagner, D.: Implementations of routing algorithms for transportation networks. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths. DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
4. Bast, H., Funke, S., Matijevic, D.: Transit: Ultrafast shortest-path queries with linear-time preprocessing. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
5. Delling, D., Holzer, M., Muller, K., Schulz, F., Wagner, D.: High-performance multi-level graphs. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
6. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Highway hierarchies star. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
7. Dijkstra, E.: A note on two problems in connexion with graphs. Numerische Mathematik **1**, 269–271 (1959)
8. Edmonds, N., Breuer, A., Gregor, D., Lumsdaine, A.: Single-source shortest paths with the parallel boost graph library. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
9. Goldberg, A., Kaplan, H., Werneck, R.: Better landmarks within reach. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths. DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
10. Köhler, E., Möhring, R., Schilling, H.: Fast point-to-point shortest path computations with arc-flags. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
11. Lauther, U.: An experimental evaluation of point-to-point shortest path calculation on roadnetworks with precalculated edge-flags. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
12. Madduri, K., Bader, D., Berry, J., Crobak, J.: Parallel shortest path algorithms for solving large-scale instances. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
13. Pascoal, M.: Implementations and empirical comparison of k shortest loopless path algorithms. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
14. Sanders, P., Schultes, D.: Robust, almost constant time shortest-path queries in road networks. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
15. Santos, J.: K shortest path algorithms. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006

Implementation Challenge for TSP Heuristics

2002; Johnson, McGeoch

Lyle A. McGeoch
Department of Mathematics and Computer Science, Amherst College, Amherst, MA, USA

Keywords and Synonyms

Lin-Kernighan; Two-opt; Three-opt; Held-Karp; TSPLIB; Concorde

Problem Definition

The Eighth DIMACS Implementation Challenge, sponsored by DIMACS, the Center for Discrete Mathematics and Theoretical Computer Science, concerned heuristics for the symmetric Traveling Salesman Problem. The Challenge began in June 2000 and was organized by David S. Johnson, Lyle A. McGeoch, Fred Glover and César Rego. It explored the state-of-the-art in the area of TSP heuristics, with researchers testing a wide range of implementations on a common (and diverse) set of input instances. The Challenge remained ongoing in 2007, with new results still being accepted by the organizers and posted on the Challenge website: www.research.att.com/~dsj/chtsp. A summary of the submissions through 2002 appeared in a book chapter by Johnson and McGeoch [5].

Participants tested their heuristics on four types of instances, chosen to test the robustness and scalability of different approaches:

1. The 34 instances that have at least 1000 cities in TSPLIB, the instance library maintained by Gerd Reinelt.
2. A set of 26 instances consisting of points uniformly distributed in the unit square, with sizes ranging from 1000 to 10,000,000 cities.
3. A set of 23 randomly generated clustered instances, with sizes ranging from 1000 to 316,000 cities.
4. A set of 7 instances based on random distance matrices, with sizes ranging from 1000 to 10,000 cities.

The TSPLIB instances and generators for the random instances are available on the Challenge website. In addition, the website contains a collection of instances for the asymmetric TSP problem.

For each instance upon which a heuristic was tested, the implementers reported the machine used, the tour length produced, the user time, and (if possible) memory usage. Some heuristics could not be applied to all of the instances, either because the heuristics were inherently geometric or because the instances were too large. To help facilitate timing comparisons between heuristics tested on different machines, participants ran a benchmark heuristic (provided by the organizers) on instances of different sizes. The benchmark times could then be used to normalize, at least approximately, the observed running times of the participants' heuristics.

The quality of a tour was computed from a submitted tour length in two ways: as a ratio over the optimal tour length for the instance (if known), and as a ratio over the Held-Karp (HK) lower bound for the instance. The Concorde optimization package of Applegate et al. [1] was able to find the optimum for 58 of the instances in reasonable time. Concorde was used in a second way to compute the HK lower bound for all but the three largest instances. A third algorithm, based on Lagrangian relaxation, was used to compute an approximate HK bound, a lower bound on true HK bound, for the remaining instances. The Challenge website reports on each of these three algorithms, presenting running times and a comparison of the bounds obtained for each instance.

The Challenge website permits a variety of reports to be created:

1. For each heuristic, tables can be generated with results for each instance, including tour length, tour quality, and raw and normalized running times.
2. For each instance, a table can be produced showing the tour quality and normalized running time of each heuristic.
3. For each pair of heuristics, tables and graphs can be produced that compare tour quality and running time for instances of different type and size.

Heuristics for which results were submitted to the Challenge fell into several broad categories:

Heuristics designed for speed. These heuristics – all of which target geometric instances – have running times within a small multiple of the time needed to read the input instance. Examples include the strip and spacefilling-curve heuristics. The speed requirement affects tour quality dramatically. Two of these algorithms produced tours with 14% of the HK lower bound for a particular TSPLIB instance, but none came within 25% on the other 89 instances.

Tour construction heuristics. These heuristics construct tours in various ways, without seeking to find improvements once a single tour passing through all cities is found. Some are simple, such as the nearest-neighbor and greedy heuristics, while others are more complex, such as the famous Christofides heuristic. These heuristics offer a number of options in trading time for tour quality, and several produce tours within 15% of the HK lower bound on most instances in reasonable time. The best of them, a variant of Christofides, produces tours within 8% on uniform instances but is much more time-consuming than the other algorithms.

Simple local improvement heuristics. These include the well-known two-opt and three-opt heuristics and variants of them. These heuristics outperform tour construction heuristics in terms of tour quality on most types of instances. For example, 3-opt gets within about 3% of the HK lower bound on most uniform instances. The submissions in this category explored various implementation choices that affect the time-quality tradeoff.

Lin-Kernighan and its variants. These heuristics extend the local search neighborhood used in 3-opt. Lin-Kernighan can produce high-quality tours (for example, within 2% of the HK lower bound on uniform instances) in reasonable time. One variant, due to Helsgaun [3], obtains tours within 1% on a wide variety of instances, although the running time can be substantial.

Repeated local search heuristics. These heuristics are based on repeated executions of a heuristic such as Lin-Kernighan, with random kicks applied to the tour after a local optimum is found. These algorithms can yield high-quality tours at increased running time.

Heuristics that begin with repeated local search. One example is the tour-merge heuristic [2], which runs repeated local search multiple times, builds a graph containing edges found in the best tours, and does exhaustive search within the resulting graph. This approach yields the best known tours for some of the instances in the Challenge.

The submissions to the Challenge demonstrated the remarkable effectiveness of heuristics for the traveling salesman problem. They also showed that implementation details, such a choice of data structure or whether to approximate aspects of the computation, can affect running time and/or solution quality greatly. Results for a given heuristic also varied enormously depending on the type of instance to which it is applied.

URL to Code

www.research.att.com/~dsj/chtsp

Cross References

► TSP-Based Curve Reconstruction

Recommended Reading

1. Applegate, D., Bixby, R., Chvátal, V., Cook, W.: On the solution of traveling salesman problems. Documenta Mathematica, Extra Volume Proceedings ICM III:645–656. Deutsche Mathematiker-Vereinigung, Berlin (1998)
2. Applegate, D., Bixby, R., Chvátal, V., Cook, W.: Finding tours in the TSP. Technical Report 99885, Research Institute for Discrete Mathematics, Universität Bonn (1999)
3. Helsgaun, K.: An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur. J. Oper. Res. **126**(1), 106–130 (2000)
4. Johnson, D.S., McGeoch, L.A.: The traveling salesman problem: A case study. In: Aarts, E., Lenstra, J.K. (eds.) Local Search in Combinatorial Optimization, pp. 215–310. Wiley, Chicester (1997)
5. Johnson, D.S., McGeoch, L.A.: Experimental analysis of heuristics for the STSP. In: Gutin, G., Punnen, A.P. (eds.) The Traveling Salesman Problem and Its Variants, pp. 369–443, Kluwer, Dordrecht (2002)

Implementing Shared Registers in Asynchronous Message-Passing Systems

1995; Attiya, Bar-Noy, Dolev

Eric Ruppert
Department Computer Science and Engineering, York University, Toronto, ON, Canada

Keywords and Synonyms

Simulation; Emulation

Problem Definition

A distributed system is composed of a collection of n processes which communicate with one another. Two means of interprocess communication have been heavily studied. *Message-passing systems* model computer networks where each process can send information over message channels to other processes. In *shared-memory systems*, processes communicate less directly by accessing information in shared data structures. Distributed algorithms are often easier to design for shared-memory systems because of their similarity to single-process system architectures. However, many real distributed systems are constructed as message-passing systems. Thus, a key problem in distributed computing is the implementation of shared memory in message-passing systems. Such implementations are also called simulations or emulations of shared memory.

The most fundamental type of shared data structure to implement is a *(read-write) register*, which stores a value, taken from some domain D. It is initially assigned a value from D and can be accessed by two kinds of operations, read and write(v), where $v \in D$. A register may be either *single-writer*, meaning only one process is allowed to write it, or *multi-writer*, meaning any process may write to it. Similarly, it may be either *single-reader* or *multi-reader*. Attiya and Welch [4] give a survey of how to build multi-writer, multi-reader registers from single-writer, single-reader ones.

If reads and writes are performed one at a time, they have the following effects: a read returns the value stored in the register to the invoking process, and a write(v) changes the value stored in the register to v and returns an acknowledgment, indicating that the operation is complete. When many processes apply operations concurrently, there are several ways to specify a register's behavior [14]. A single-writer register is *regular* if each read returns either the argument of the write that completed most recently before the read began or the argument of some

write operation that runs concurrently with the read. (If there is no write that completes before the read begins, the read may return either the initial value of the register or the value of a concurrent write operation.) A register is *atomic* (see ► linearizability) if each operation appears to take place instantaneously. More precisely, for any concurrent execution, there is a total order of the operations such that each read returns the value written by the last write that precedes it in the order (or the initial value of the register, if there is no such write). Moreover, this total order must be consistent with the temporal order of operations: if one operation finishes before another one begins, the former must precede the latter in the total order. Atomicity is a stronger condition than regularity, but it is possible to implement atomic registers from regular ones with some complexity overhead [12].

This article describes the problem of implementing registers in an asynchronous message-passing system in which processes may experience crash failures. Each process can send a message, containing a finite string, to any other process. To make the descriptions of algorithms more uniform, it is often assumed that processes can send messages to themselves. All messages are eventually delivered. In the algorithms described below, senders wait for an acknowledgment of each message before sending the next message, so it is not necessary to assume that the message channels are first-in-first-out. The system is totally asynchronous: there is no bound on the time required for a message to be delivered to its recipient or for a process to perform a step of local computation. A process that fails by crashing stops executing its code, but other processes cannot distinguish between a process that has crashed and one that is running very slowly. (Failures of message channels [3] and more malicious kinds of process failures [15] have also been studied.)

A *t-resilient* register implementation provides programmes to be executed by processes to simulate read and write operations. These programmes can include any standard control structures and accesses to a process's local memory, as well as instructions to send a message to another process and to read the process's buffer, where incoming messages are stored. The implementation should also specify how the processes' local variables are initialized to reflect any initial value of the implemented register. In the case of a single-writer register, only one process may execute the write programme. A process may invoke the read and write programmes repeatedly, but it must wait for one invocation to complete before starting the next one. In any such execution where at most t processes crash, each of a process's invocations of the read or write programme should eventually terminate. Each read operation returns a result from the set D, and these results should satisfy regularity or atomicity.

Relevant measures of algorithm complexity include the number of messages transmitted in the system to perform an operation, the number of bits per message, and the amount of local memory required at each process. One measure of time complexity is the time needed to perform an operation, under the optimistic assumption that the time to deliver messages is bounded by Δ and local computation is instantaneous (although algorithms must work correctly even without these assumptions).

Key Results

Implementing a Regular Register

One of the core ideas for implementing shared registers in message-passing systems is a construction that implements a regular single-writer multi-reader register. It was introduced by Attiya, Bar-Noy and Dolev [3] and made more explicit by Attiya [2]. A write(v) sends the value v to all processes and waits until a majority of the processes ($\lfloor \frac{n}{2} \rfloor + 1$, including the writer itself) return an acknowledgment. A reader sends a request to all processes for their latest values. When it has received responses from a majority of processes, it picks the most recently written value among them. If a write completes before a read begins, at least one process that answers the reader has received the write's value prior to sending its response to the reader. This is because any two sets that each contain a majority of the processes must overlap. The time required by operations when delivery times are bounded is 2Δ.

This algorithm requires the reader to determine which of the values it receives is most recent. It does this using *timestamps* attached to the values. If the writer uses increasing integers as timestamps, the messages grow without bound as the algorithm runs. Using the bounded timestamp scheme of Israeli and Li [13] instead yields the following theorem.

Theorem 1 (Attiya [2]) *There is an $\lceil \frac{n-2}{2} \rceil$-resilient implementation of a regular single-writer, multi-reader register in a message-passing system of n processes. The implementation uses $\Theta(n)$ messages per operation, with $\Theta(n^3)$ bits per message. The writer uses $\Theta(n^4)$ bits of local memory and each reader uses $\Theta(n^3)$ bits.*

Theorem 1 is optimal in terms of fault-tolerance. If $\lceil \frac{n}{2} \rceil$ processes can crash, the network can be partitioned into two halves of size $\lfloor \frac{n}{2} \rfloor$, with messages between the two halves delayed indefinitely. A write must terminate before any evidence of the write is propagated to the half not containing the writer, and then a read performed by

a process in that half cannot return an up-to-date value. For $t \geq \lceil \frac{n}{2} \rceil$, registers can be implemented in a message-passing system only if some degree of synchrony is present in the system. The exact amount of synchrony required was studied by Delporte-Gallet et al. [6].

Theorem 1 is within a constant factor of the optimal number of messages per operation. Evidence of each write must be transmitted to at least $\lceil \frac{n}{2} \rceil - 1$ processes, requiring $\Omega(n)$ messages; otherwise this evidence could be obliterated by crashes. A write must terminate even if only $\lfloor \frac{n}{2} \rfloor + 1$ processes (including the writer) have received information about the value written, since the rest of the processes could have crashed. Thus, a read must receive information from at least $\lceil \frac{n}{2} \rceil$ processes (including itself) to ensure that it is aware of the most recent write operation.

A t-resilient implementation, for $t < \lceil \frac{n}{2} \rceil$, that uses $\Theta(t)$ messages per operation is obtained by the following adaptation. A set of $2t + 1$ processes is preselected to be data storage servers. Writes send information to the servers, and wait for $t + 1$ acknowledgments. Reads wait for responses from $t + 1$ of the servers and choose the one with the latest timestamp.

Implementing an Atomic Register

Attiya, Bar-Noy and Dolev [3] gave a construction of an atomic register in which readers forward the value they return to all processes and wait for an acknowledgment from a majority. This is done to ensure that a read does not return an older value than another read that precedes it. Using unbounded integer timestamps, this algorithm uses $\Theta(n)$ messages per operation. The time needed per operation when delivery times are bounded is 2Δ for writes and 4Δ for reads. However, their technique of bounding the timestamps increases the number of messages per operation to $\Theta(n^2)$ (and the time per operation to 12Δ). A better implementation of atomic registers with bounded message size is given by Attiya [2]. It uses the regular registers of Theorem 1 to implement atomic registers using the "handshaking" construction of Haldar and Vidyasankar [12], yielding the following result.

Theorem 2 (Attiya [2]) *There is an $\lceil \frac{n-2}{2} \rceil$-resilient implementation of an atomic single-writer, multi-reader register in a message-passing system of n processes. The implementation uses $\Theta(n)$ messages per operation, with $\Theta(n^3)$ bits per message. The writer uses $\Theta(n^5)$ bits of local memory and each reader uses $\Theta(n^4)$ bits.*

Since atomic registers are regular, this algorithm is optimal in terms of fault-tolerance and within a constant factor of optimal in terms of the number of messages. The time used when delivery times are bounded is at most 14Δ for writes and 18Δ for reads.

Applications

Any distributed algorithm that uses shared registers can be adapted to run in a message-passing system using the implementations described above. This approach yielded new or improved message-passing solutions for a number of problems, including randomized consensus [1], multi-writer registers [4], and snapshot objects ▶ Snapshots. The reverse simulation is also possible, using a straightforward implementation of message channels by single-writer, single-reader registers. Thus, the two asynchronous models are equivalent, in terms of the set of problems that they can solve, assuming only a minority of processes crash. However there is some complexity overhead in using the simulations.

If a shared-memory algorithm is implemented in a message-passing system using the algorithms described here, processes must continue to operate even when the algorithm terminates, to help other processes execute their reads and writes. This cannot be avoided: if each process must stop taking steps when its algorithm terminates, there are some problems solvable with shared registers that are not solvable in the message-passing model [5].

Using a majority of processes to "validate" each read and write operation is an example of a quorum system, originally introduced for replicated data by Gifford [10]. In general, a quorum system is a collection of sets of processes, called quorums, such that every two quorums intersect. Quorum systems can also be designed to implement shared registers in other models of message-passing systems, including dynamic networks and systems with malicious failures. For examples, see [7,9,11,15].

Open Problems

Although the algorithms described here are optimal in terms of fault-tolerance and message complexity, it is not known if the number of bits used in messages and local memory is optimal. The exact time needed to do reads and writes when messages are delivered within time Δ is also a topic of ongoing research. (See, for example, [8].) As mentioned above, the simulation of shared registers can be used to implement shared-memory algorithms in message-passing systems. However, because the simulation introduces considerable overhead, it is possible that some of those problems could be solved more efficiently by algorithms designed specifically for message-passing systems.

Cross References

- ▶ Linearizability
- ▶ Quorums
- ▶ Registers

Recommended Reading

1. Aspnes, J.: Randomized protocols for asynchronous consensus. Distrib. Comput. **16**(2–3), 165–175 (2003)
2. Attiya, H.: Efficient and robust sharing of memory in message-passing systems. J. Algorithms **34**(1), 109–127 (2000)
3. Attiya, H., Bar-Noy, A., Dolev, D.: Sharing memory robustly in message-passing systems. J. ACM **42**(1), 124–142 (1995)
4. Attiya, H., Welch, J.: Distributed Computing: Fundamentals, Simulations and Advanced Topics, 2nd edn. Wiley-Interscience, Hoboken (2004)
5. Chor, B., Moscovici, L.: Solvability in asynchronous environments. In: Proc. 30th Symposium on Foundations of Computer Science, pp. 422–427 (1989)
6. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Hadzilacos, V., Kouznetsov, P., Toueg, S.: The weakest failure detectors to solve certain fundamental problems in distributed computing. In: Proc. 23rd ACM Symposium on Principles of Distributed Computing, pp. 338–346. St. John's, Newfoundland, 25–28 July 2004
7. Dolev, S., Gilbert, S., Lynch, N.A., Shvartsman, A.A., Welch, J.L.: GeoQuorums: Implementing atomic memory in mobile ad hoc networks. Distrib. Comput. **18**(2), 125–155 (2005)
8. Dutta, P., Guerraoui, R., Levy, R.R., Chakraborty, A.: How fast can a distributed atomic read be? In: Proc. 23rd ACM Symposium on Principles of Distributed Computing, pp. 236–245. St. John's, Newfoundland, 25–28 July 2004
9. Englert, B., Shvartsman, A.A.: Graceful quorum reconfiguration in a robust emulation of shared memory. In: Proc. 20th IEEE International Conference on Distributed Computing Systems, pp. 454–463. Taipei, 10–13 April 2000
10. Gifford, D.K.: Weighted voting for replicated data. In: Proc. 7th ACM Symposium on Operating Systems Principles, pp. 150–162. Pacific Grove, 10–12 December 1979
11. Gilbert, S., Lynch, N., Shvartsman, A.: Rambo II: rapidly reconfigurable atomic memory for dynamic networks. In: Proc. International Conference on Dependable Systems and Networks, pp. 259–268. San Francisco, 22–25 June 2003
12. Haldar, S., Vidyasankar, K.: Constructing 1-writer multireader multivalued atomic variables from regular variables. J. ACM **42**(1), 186–203 (1995)
13. Israeli, A., Li, M.: Bounded time-stamps. Distrib. Comput. **6**(4), 205–209 (1993)
14. Lamport, L.: On interprocess communication, Part II: Algorithms. Distrib. Comput. **1**(2), 86–101 (1986)
15. Malkhi, D., Reiter, M.: Byzantine quorum systems. Distrib. Comput. **11**(4), 203–213 (1998)

Incentive Compatible Algorithms

- ▶ Computing Pure Equilibria in the Game of Parallel Links
- ▶ Truthful Mechanisms for One-Parameter Agents

Incentive Compatible Selection

2006; Chen, Deng, Liu

Xi Chen[1], Xiaotie Deng[2]

[1] Department of Computer Science and Technology, Tsinghua University, Beijing, China

[2] Department of Computer Science, City University of Hong Kong, Hong Kong, China

Keywords and Synonyms

Incentive compatible selection; Incentive compatible ranking; Algorithmic mechanism design

Problem Definition

Ensuring truthful evaluation of alternatives in human activities has always been an important issue throughout history. In sport, in particular, such an issue is vital and practice of the fair play principle has been consistently put forward as a matter of foremost priority. In addition to relying on the code of ethics and professional responsibility of players and coaches, the design of game rules is an important measure in enforcing fair play.

Ranking alternatives through pairwise comparisons (or competitions) is the most common approach in sports tournaments. Its goal is to find out the "true" ordering among alternatives through complete or partial pairwise competitions [1, 3, 4, 5, 6, 7]. Such studies have been mainly based on the assumption that all the players play truthfully, i. e., with their maximal effort. It is, however, possible that some players form a coalition, and cheat for group benefit. An interesting example can be found in [2].

Problem Description

The work of Chen, Deng, and Liu [2] considers the problem of choosing m winners out of n candidates.

Suppose a tournament is held among n players $P_n = \{p_1, \ldots p_n\}$ and m winners are expected to be selected by a selection protocol. Here a protocol $f_{n,m}$ is a predefined function (which will become clear later) to choose winners through pairwise competitions, with the intention of finding m players of highest capacity. When the tournament starts, a distinct ID in $N_n = \{1, 2, \ldots n\}$ is assigned to each player in P_n by a randomly picked indexing function $I : P_n \to N_n$. Then a match is played between each pair of players. The competition outcomes will form a graph G, whose vertex set is N_n and edges represent the results of all the matches. Finally, the graph will be treated as the input to $f_{n,m}$, and it will output a set of m winners. Now it

should be clear that $f_{n,m}$ maps every possible tournament graph G to a subset (of cardinality m) of N_n.

Suppose there exists a group of bad players who play dishonestly, i. e. they might lose a match on purpose to gain overall benefit for the whole group, while the rest of the players always play truthfully, i. e. they try their best to win matches. The group of bad players gains benefit if they are able to have more winning positions than that according to the true ranking. Given knowledge of the selection protocol $f_{n,m}$, the indexing function I and the true ranking of all players, the bad players try to find a cheating strategy that can fool the protocol and gain benefit.

The problem is discussed under two models in which the characterizations of bad players are different. Under the *collective incentive compatible model*, bad players are willing to sacrifice themselves to win group benefit; while the ones under the *alliance incentive compatible model* only cooperate if their individual interests are well maintained in the cheating strategy.

The goal is to find an "ideal" protocol, under which players or groups of players maximize their benefits only by strictly following the fair play principle, i. e. always play with maximal effort.

Formal Definitions

When the tournament begins, an indexing function I is randomly picked, which assigns ID $I(p) \in N_n$ to each player $p \in P_n$. Then a match is played between each pair of players, and the results are represented as a directed graph G. Finally, G is fed into the predefined selection protocol $f_{n,m}$, to produce a set of m winners $I^{-1}(W)$, where $W = f_{n,m}(G) \subset N_n$.

Notations An indexing function I for a tournament attended by n players $P_n = \{p_1, p_2, \dots p_n\}$ is a one-to-one correspondence from P_n to the set of *IDs*: $N_n = \{1, 2, \dots n\}$. A ranking function R is a one-to-one correspondence from P_n to $\{1, 2, \dots n\}$. $R(p)$ represents the underlying true ranking of player p among the n players. The smaller, the stronger.

A tournament graph of size n is a directed graph $G = (N_n, E)$ such that, for all $i \neq j \in N_n$, either $ij \in E$ (player with ID i beats player with ID j) or $ji \in E_n$. Let K_n denote the set of all such graphs. A selection protocol $f_{n,m}$, which chooses m winners out of n candidates, is a function from K_n to $\{S \subset N_n \text{ and } |S| = m\}$.

A tournament T_n among players P_n is a pair $T_n = (R, B)$ where R is a ranking function from P_n to N_n and $B \subset P_n$ is the group of bad players.

Definition 1 (Benefit) Given a protocol $f_{n,m}$, a tournament $T_n = (R, B)$, an indexing function I and a tournament graph $G \in K_n$, the benefit of the group of bad players is

$$\mathbf{Ben}(f_{n,m}, T_n, I, G) = \left|\left\{i \in f_{n,m}(G),\ I^{-1}(i) \in B\right\}\right| - \left|\left\{p \in B,\ R(p) \leq m\right\}\right|.$$

Given knowledge of $f_{n,m}$, T_n and I, not every $G \in K_n$ is a feasible strategy for B: the group of bad players. First, it depends on the tournament $T_n = (R, B)$, e. g. a player $p_b \in B$ cannot win a player $p_g \notin B$ if $R(p_b) > R(p_g)$. Second, it depends on the property of bad players which is specified by the model considered. Tournament graphs, which are recognized as feasible strategies, are characterized below, for each model. The key difference is that, a bad player in the alliance incentive compatible model is not willing to sacrifice his own winning position, while a player in the other model fights for group benefit at all costs.

Definition 2 (Feasible Strategy) Given $f_{n,m}$, $T_n = (R, B)$ and I, graph $G \in K_n$ is *c-feasible* if

1 For every two players $p_i, p_j \notin B$, if $R(p_i) < R(p_j)$, then $I(p_i)I(p_j) \in E$;
2 For all $p_g \notin B$ and $p_b \in B$, if $R(p_g) < R(p_b)$, then edge $I(p_g)I(p_b) \in E$.

Graph $G \in K_n$ is *a-feasible* if it is c-feasible and also satisfies

3 For every bad player $p \in B$, if $R(p) \leq m$, then $I(p) \in f_{n,m}(G)$.

A cheating strategy is then a feasible tournament graph G that can be employed by the group of bad players to gain positive benefit.

Definition 3 (Cheating Strategy) Given $f_{n,m}$, $T_n = (R, B)$ and I, a cheating strategy for the group of bad players under the collective incentive compatible (alliance incentive compatible) model is a graph $G \in K_n$ which is c-feasible (a-feasible) and satisfies $\boldsymbol{Ben}(f_{n,m}, T_n, I, G) > 0$.

The following two problems are studied in [2]: (**1**) Is there a protocol $f_{n,m}$ such that for all T_n and I, no cheating strategy exists under the collective incentive compatible model? (**2**) Is there a protocol $f_{n,m}$ such that for all T_n and I, no cheating strategy exists under the alliance incentive compatible model?

Key Results

Definition 4 For all integers n and m such that $2 \leq m \leq n - 2$, a tournament graph $G_{n,m} = (N_n, E) \in K_n$, which consists of three parts T_1, T_2, and T_3, is defined as follows:

1 $T_1 = \{1, 2, \ldots m-2\}$. For all $i < j \in T_1$, edge $ij \in E$;
2 $T_2 = \{m-1, m, m+1\}$. $(m-1)m$, $m(m+1)$, $(m+1)(m-1) \in E$;
3 $T_3 = \{m+2, m+3, \ldots n\}$. For all $i < j \in T_3$, edge $ij \in E$;
4 For all $i' \in T_i$ and $j' \in T_j$ such that $i < j$, edge $i'j' \in E$.

Theorem 1 *Under the collective incentive compatible model, for every selection protocol $f_{n,m}$ with $2 \le m \le n-2$, if $T_n = (R, B)$ satisfies:* **(1)** *At least one bad player ranks as high as $m-1$;* **(2)** *The ones ranked $m+1$ and $m+2$ are both bad players;* **(3)** *The one ranked m is a good player, then there always exists an indexing function I such that $G_{n,m}$ is a cheating strategy.*

Theorem 2 *Under the alliance incentive compatible model, if $n - m \ge 3$, then there exists a selection protocol $f_{n,m}$ [2] such that, for every tournament T_n, indexing function I and a-feasible strategy $G \in K_n$,* $\mathbf{Ben}(f_{n,m}, T_n, I, G) \le 0$.

Applications

The result shows that, if players are willing to sacrifice themselves, no protocol is able to prevent malicious coalitions from obtaining undeserved benefits.

The result may have potential applications in the design of output truthful mechanisms.

Open Problems

Under the collective incentive compatible model, the work of Chen, Deng, and Liu indicates that cheating strategies are available in at least 1/8 tournaments, assuming the probability for each player to be in the bad group is 1/2. Could this bound be improved? Or could one find a good selection protocol in the sense that the number of tournaments with cheating strategies is close to this bound? On the other hand, although no ideal protocol exists in this model, does there exist any randomized protocol, under which the probability of having cheating strategies is negligible?

Cross References

► Algorithmic Mechanism Design
► Truthful Multicast

Recommended Reading

1. Chang, P., Mendonca, D., Yao, X., Raghavachari, M.: An evaluation of ranking methods for multiple incomplete round-robin tournaments. In: Proceedings of the 35th Annual Meeting of Decision Sciences Institute, Boston, 20–23 November 2004
2. Chen, X., Deng, X., Liu, B.J.: On incentive compatible competitive selection protocol. In: COCOON'06: Proceedings of the 12th Annual International Computing and Combinatorics Conference, pp. 13–22, Taipei, 15–18 August 2006
3. Harary, F., Moser, L.: The theory of round robin tournaments. Am. Math. Mon. **73**(3), 231–246 (1966)
4. Jech, T.: The ranking of incomplete tournaments: A mathematician's guide to popular sports. Am. Math. Mon. **90**(4), 246–266 (1983)
5. Mendonca, D., Raghavachari, M.: Comparing the efficacy of ranking methods for multiple round-robin tournaments. Eur. J. Oper. Res. **123**, 593–605 (1999)
6. Rubinstein, A.: Ranking the participants in a tournament. SIAM J. Appl. Math. **38**(1), 108–111 (1980)
7. Steinhaus, H.: Mathematical Snapshots. Oxford University Press, New York (1950)

Incremental Algorithms

► Fully Dynamic Connectivity
► Fully Dynamic Transitive Closure

Independent Sets in Random Intersection Graphs
2004; Nikoletseas, Raptopoulos, Spirakis

Sotiris Nikoletseas, Christoforos Raptopoulos, Paul Spirakis
Research Academic Computer Technology Institute, Greece and Computer Engineering and Informatics Department, University of Patras, Patras, Greece

Keywords and Synonyms

Existence and efficient construction of independent sets of vertices in general random intersection graphs

Problem Definition

This problem is concerned with the efficient construction of an independent set of vertices (i. e. a set of vertices with no edges between them) with maximum cardinality, when the input is an instance of the uniform random intersection graphs model. This model was introduced by Karoński, Sheinerman, and Singer-Cohen in [4] and Singer-Cohen in [10] and it is defined as follows

Definition 1 (Uniform random intersection graph) Consider a universe $M = \{1, 2, \ldots, m\}$ of elements and a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$. If one assigns independently to each vertex v_j, $j = 1, 2, \ldots, n$, a subset S_{v_j} of M by choosing each element independently with probability p and puts an edge between two vertices v_{j_1}, v_{j_2} if and

only if $S_{v_{j_1}} \cap S_{v_{j_2}} \neq \emptyset$, then the resulting graph is an instance of the uniform random intersection graph $G_{n,m,p}$.

The universe M is sometimes called *label set* and its elements *labels*. Also, denote by L_l, for $l \in M$, the set of vertices that have chosen label l.

Because of the dependence of edges, this model can abstract more accurately (than the Bernoulli random graphs model $G_{n,p}$ that assumes independence of edges) many real-life applications. Furthermore, Fill, Sheinerman, and Singer-Cohen show in [3] that for some ranges of the parameters n, m, p ($m = n^\alpha, \alpha > 6$), the spaces $G_{n,m,p}$ and $G_{n,\hat{p}}$ are equivalent in the sense that the total variation distance between the graph random variables has limit 0. The work of Nikoletseas, Raptopoulos, and Spirakis [7] introduces two new models, namely the *general random intersection graphs model* $G_{n,m,\vec{p}}$, $\vec{p} = [p_1, p_2, \ldots, p_m]$ and the *regular random intersection graphs model* $G_{n,m,\lambda}, \lambda > 0$ that use a different way to randomly assign labels to vertices, but the edge appearance rule remains the same. The $G_{n,m,\vec{p}}$ model is a generalization of the uniform model where each label $i \in M$ is chosen independently with probability p_i, whereas in the $G_{n,m,\lambda}$ model each vertex chooses a random subset of M with exactly λ labels.

The authors in [7] first consider the existence of independent sets of vertices of a given cardinality in general random intersection graphs and provide exact formulae for the mean and variance of the number of independent sets of vertices of cardinality k. Furthermore, they present and analyze three polynomial time (on the number of labels m and the number of vertices n) algorithms for constructing large independent sets of vertices when the input is an instance of the $G_{n,m,p}$ model. To the best knowledge of the entry authors, this work is the first to consider algorithmic issues for these models of random graphs.

Key Results

The following theorems concern the existence of independent sets of vertices of cardinality k in general random intersection graphs. The proof of Theorem 1 uses the linearity of expectation of sums of random variables.

Theorem 1 *Let $X^{(k)}$ denote the number of independent sets of size k in a random intersection graph $G(n, m, \vec{p})$, where $\vec{p} = [p_1, p_2, \ldots, p_m]$. Then*

$$E\left[X^{(k)}\right] = \binom{n}{k} \prod_{i=1}^{m} \left((1-p_i)^k + k p_i (1-p_i)^{k-1}\right) .$$

Theorem 2 *Let $X^{(k)}$ denote the number of independent sets of size k in a random intersection graph $G(n, m, \vec{p})$, where $\vec{p} = [p_1, p_2, \ldots, p_m]$. Then*

$$\mathrm{Var}\left(X^{(k)}\right) = \sum_{s=1}^{k} \binom{n}{2k-s} \binom{2k-s}{s} \left(\gamma(k,s) \frac{E\left[X^{(k)}\right]}{\binom{n}{k}} - \frac{E^2\left[X^{(k)}\right]}{\binom{n}{k}^2}\right)$$

where $E\left[X^{(k)}\right]$ is the mean number of independent sets of size k and

$$\gamma(k,s) = \prod_{i=1}^{m} \left((1-p_i)^{k-s} + (k-s) p_i (1-p_i)^{k-s-1} \left(1 - \frac{s p_i}{1+(k-1)p_i}\right)\right) .$$

Theorem 2 is proved by first writing the variance as the sum of covariances and then applying a vertex contraction technique that merges several vertices into one supervertex with similar probabilistic behavior in order to compute the covariances. By using the second moment method (see [1]) one can derive thresholds for the existence of independent sets of size k.

One of the three algorithms that were proposed in [7] is presented below. The algorithm starts with V (i. e. the set of vertices of the graph) as its "candidate" independent set. In every subsequent step it chooses a label and removes from the current candidate independent set all vertices having that label in their assigned label set except for one. Because of the edge appearance rule, this ensures that after doing this for every label in M, the final candidate independent set will contain only vertices that do not have edges between them and so it will be indeed an independent set.

Algorithm:
Input: A random intersection graph $G_{n,m,p}$.
Output: An independent set of vertices A_m.
1. set $A_0 := V$; set $L := M$;
2. **for** $i = 1$ **to** m **do**
3. **begin**
4. select a random label $l_i \in L$; set $L := L - \{l_i\}$;
5. set $D_i := \{v \in A_{i-1} : l_i \in S_v\}$;
6. **if** $(|D_i| \geq 1)$ **then** select a random vertex $u \in D_i$ and set $D_i := D_i - \{u\}$;
7. set $A_i := A_{i-1} - D_i$;
8. **end**
9. **output** A_m;

The following theorem concerns the cardinality of the independent set produced by the algorithm. The analysis of the algorithm uses Wald's equation (see [9]) for sums

of a random number of random variables to calculate the mean value of $|A_m|$, and also Chernoff bounds (see e. g. [6]) for concentration around the mean.

Theorem 3 *For the case $mp = \alpha \log n$, for some constant $\alpha > 1$ and $m \geq n$, and for some constant $\beta > 0$, the following hold with high probability:*

1. *If $np \to \infty$ then $|A_m| \geq (1-\beta)\frac{n}{\log n}$.*
2. *If $np \to b$ where $b > 0$ is a constant then $|A_m| \geq (1-\beta)n(1-\mathrm{e}^{-b})$.*
3. *If $np \to 0$ then $|A_m| \geq (1-\beta)n$.*

The above theorem shows that the algorithm manages to construct a quite large independent set with high probability.

Applications

First of all, note that (as proved in [5]) any graph can be transformed into an intersection graph. Thus, the random intersection graphs models can be very general. Furthermore, for some ranges of the parameters n, m, p ($m = n^\alpha, \alpha > 6$) the spaces $G_{n,m,p}$ and $G_{n,p}$ are equivalent (as proved by Fill, Sheinerman, and Singer-Cohen in [3], showing that in this range the total variation distance between the graph random variables has limit 0).

Second, random intersection graphs (and in particular the general intersection graphs model of [7]) may model real-life applications more accurately (compared to the $G_{n,p}$ case). In particular, such graphs can model resource allocation in networks, e. g. when network nodes (abstracted by vertices) access shared resources (abstracted by labels): the intersection graph is in fact the conflict graph of such resource allocation problems.

Other Related Work

In their work [4] Karoński et al. consider the problem of the emergence of graphs with a constant number of vertices as induced subgraphs of $G_{n,m,p}$ graphs. By observing that the $G_{n,m,p}$ model generates graphs via clique covers (for example the sets $L_l, l \in M$ constitute an obvious clique cover) they devise a natural way to use them together with the first and second moment methods in order to find thresholds for the appearance of any fixed graph H as an induced subgraph of $G_{n,m,p}$ for various values of the parameters n, m and p.

The connectivity threshold for $G_{n,m,p}$ was considered by Singer-Cohen in [10]. She studies the case $m = n^\alpha, \alpha > 0$ and distinguishes two cases according to the value of α. For the case $\alpha > 1$, the results look similar to the $G_{n,p}$ graphs, as the mean number of edges at the connectivity thresholds are (roughly) the same. On the other hand, for $\alpha \leq 1$ we get denser graphs in the $G_{n,m,p}$ model. Besides connectivity, [10] examines also the size of the largest clique in uniform random intersection graphs for certain values of n, m and p.

The existence of Hamilton cycles in $G_{n,m,p}$ graphs was considered by Efthymiou and Spirakis in [2]. The authors use coupling arguments to show that the threshold of appearance of Hamilton cycles is quite close to the connectivity threshold of $G_{n,m,p}$. Efficient probabilistic algorithms for finding Hamilton cycles in uniform random intersection graphs were presented by Raptopoulos and Spirakis in [8]. The analysis of those algorithms verify that they perform well w.h.p. even for values of p that are close to the connectivity threshold of $G_{n,m,p}$. Furthermore, in the same work, an expected polynomial algorithm for finding Hamilton cycles in $G_{n,m,p}$ graphs with constant p is given.

In [11] Stark gives approximations of the distribution of the degree of a fixed vertex in the $G_{n,m,p}$ model. More specifically, by applying a sieve method, the author provides an exact formula for the probability generating function of the degree of some fixed vertex and then analyzes this formula for different values of the parameters n, m and p.

Open Problems

A number of problems related to random intersection graphs remain open. Nearly all the algorithms proposed so far concerning constructing large independent sets and finding Hamilton cycles in random intersection graphs are greedy. An interesting and important line of research would be to find more sophisticated algorithms for these problems that outperform the greedy ones. Also, all these algorithms were presented and analyzed in the uniform random intersection graphs model. Very little is known about how the same algorithms would perform when their input was an instance of the general or even the regular random intersection graph models.

Of course, many classical problems concerning random graphs have not yet been studied. One such example is the size of the minimum dominating set (i. e. a set of vertices that has the property that all vertices of the graph either belong to this set or are connected to it) in a random intersection graph. Also, what is the degree sequence of $G_{n,m,p}$ graphs? Note that this is very different from the problem addressed in [11].

Finally, notice that none of the results presented in the bibliography for general or uniform random intersection graphs carries over immediately to regular random intersection graphs. Of course, for some values of n, m, p and

λ, certain graph properties shown for $G_{n,m,p}$ could also be proved for $G_{n,m,\lambda}$ by showing concentration of the number of labels chosen by any vertex via Chernoff bounds. Other than that, the fixed sizes of the sets assigned to each vertex impose more dependencies to the model.

Cross References

► Hamilton Cycles in Random Intersection Graphs

Recommended Reading

1. Alon, N., Spencer, H.: The Probabilistic Method. Wiley, Inc. (2000)
2. Efthymiou, C., Spirakis, P.: On the existence of hamiltonian cycles in random intersec- tion graphs. In: Proceedings of 32st International colloquium on Automata, Languages and Programming (ICALP), pp. 690–701. Springer, Berlin Heidelberg (2005)
3. Fill, J.A., Sheinerman, E.R., Singer-Cohen, K.B.: Random intersection graphs when $m = \omega(n)$: An equivalence theorem relating the evolution of the g(n, m, p) and g(n, p) models. Random Struct. Algorithm. **16**(2), 156–176 (2000)
4. Karoński, M., Scheinerman, E.R., Singer-Cohen, K.B.: On random intersection graphs: The subgraph problem. Adv. Appl. Math. **8**, 131–159 (1999)
5. Marczewski, E.: Sur deux propriétés des classes d' ensembles. Fund. Math. **33**, 303–307 (1945)
6. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)
7. Nikoletseas, S., Raptopoulos, C., Spirakis, P.: The existence and efficient construction of large independent sets in general random intersection graphs. In: Proceedings of 31st International colloquium on Automata, Languages and Programming (ICALP), pp. 1029–1040. Springer, Berlin Heidelberg (2004) Also in the Theoretical Computer Science (TCS) Journal, accepted, to appear in 2008
8. Raptopoulos, C., Spirakis, P.: Simple and efficient greedy algorithms for hamiltonian cycles in random intersection graphs. In: Proceedings of the 16th International Symposium on Algorithms and Computation (ISAAC), pp 493–504. Springer, Berlin Heidelberg (2005)
9. Ross, S.: Stochastic Processes. Wiley (1995)
10. Singer-Cohen, K.B.: Random Intersection Graphs. Ph. D. thesis, John Hopkins University, Balimore (1995)
11. Stark, D.: The vertex degree distribution of random intersection graphs. Random Struct. Algorithms **24**, 249–258 (2004)

Indexed Approximate String Matching

2006; Chan, Lam, Sung, Tam, Wong

Wing-Kin Sung
Department of Computer Science, National University of Singapore, Singapore, Singapore

Keywords and Synonyms

Indexed inexact pattern matching problem; Indexed pattern searching problem based on hamming distance or edit distance; Indexed k-mismatch problem; Indexed k-difference problem

Problem Definition

Consider a text $S[1..n]$ over a finite alphabet Σ. One wants to build an index for S such that for any query pattern $P[1..m]$ and any integer $k \geq 0$, one can report efficiently all locations in S that match P with at most k errors. If error is measured in terms of the Hamming distance (number of character substitutions), the problem is called the k-mismatch problem. If error is measured in term of the edit distance (number of character substitutions, insertions or deletions), the problem is called the k-difference problem. The two problems are formally defined as follows.

Problem 1 (k-mismatch problem) *Consider a text $S[1..n]$ over a finite alphabet Σ. For any pattern P and threshold k, position i is an occurrence of P if the hamming distance between P and $S[i..i']$ is less than k for some i'. The k-mismatch problem asks for an index I for S such that, for any pattern P, one can report all occurrences of P in S efficiently.*

Problem 2 (k-difference problem) *Consider a text $S[1..n]$ over a finite alphabet Σ. For any pattern P and threshold k, position i is an occurrence of P if the edit distance between P and $S[i..i']$ is less than k for some i'. The k-difference problem asks for an index I for S such that, for any pattern P, one can report all occurrences of P in S efficiently.*

The major concern of the two problems is how to achieve efficient pattern searching without using a large amount of space for storing the index.

Below, assume $|\Sigma|$ (the size of the alphabet) is constant.

Key Results

Table 1 summarizes the related results in the literature. Below, briefly describes the current best results.

For indexes for exact matching ($k = 0$), the best results utilize data structures like the suffix tree, compressed suffix array, and FM-index. Theorems 1 and 2 describe those results.

Theorem 1 (Weiner, 1973 [17]) *Given a suffix tree of size $O(n)$ words, one can support exact (0-mismatch) matching in $O(m + occ)$ time where occ is the number of occurrences.*

Indexed Approximate String Matching, Table 1
Known results for k-difference matching. c is some positive constant and ϵ is some positive constant smaller than 1

Space	$k = 1$	
$O(n \log^2 n)$ words	$O(m \log n \log\log n + occ)$	[1]
$O(n \log n)$ words	$O(m \log\log n + occ)$	[2]
	$O(m + occ + \log n \log\log n)$	[8]
$O(n)$ words	$O(\min\{n, m^2\} + occ)$	[6]
	$O(m \log n + occ)$	[11]
	$O(kn^\epsilon \log n)$	[14]
	$O(n^\epsilon)$	[15]
	$O(m + occ + \log^3 n \log\log n)$	[3]
	$O(m + occ + \log n \log\log n)$	[4]
$O(n\sqrt{\log n})$ bits	$O(m \log\log n + occ)$	[12]
$O(n)$ bits	$O(m \log^2 n + occ \log n)$	[11]
	$O((m \log\log n + occ) \log^\epsilon n)$	[12]
	$O(m + (occ + \log^4 n \log\log n) \log^\epsilon n)$	[3]
$O(\|\Sigma\|n)$ words in avg	$O(m + occ)$	[13]
$O(\|\Sigma\|n)$ words	$O(m + occ)$ in avg	[13]

Space	$k = O(1)$	
$O(n \log^k n)$ words	$O(m + occ + \frac{1}{k!}(c \log n)^k \log\log n)$	[8]
$O(n \log^{k-1} n)$ words	$O(m + k^3 3^k occ + \frac{1}{k!}(c \log n)^k \log\log n)$	[3]
$O(n)$ words	$O(\min\{n, \|\Sigma\|^k m^{k+2}\} + occ)$	[6]
	$O((\|\Sigma\|m)^k \log n + occ)$	[11]
	$O(m + k^3 3^k occ + (c \log n)^{k(k+1)} \log\log n)$	[3]
	$O(\|\Sigma\|^k m^{k-1} \log n \log\log n + k^3 3^k occ)$	[4]
$O(n\sqrt{\log n})$ bits	$O((\|\Sigma\|m)^k \log\log n + occ)$	[12]
$O(n)$ bits	$O((\|\Sigma\|m)^k \log^2 n + occ \log n)$	[11]
	$O(((\|\Sigma\|m)^k \log\log n + occ) \log^\epsilon n)$	[12]
	$O(m + (k^3 3^k occ + (c \log n)^{k^2+2k} \log\log n) \log^\epsilon n)$	[3]
$O(\|\Sigma\|^k n \log^k n)$ words in avg	$O(m + occ)$	[13]
$O(\|\Sigma\|^k n \log^k n)$ words	$O(m + occ)$ in avg	[13]
$O(n \log^k n)$ words in avg	$O(3^k m^{k+1} + occ)$	[7]

Theorem 2 (Ferragina and Manzini, 2000 [9]; Grossi and Vitter [10]) *Given a compressed suffix array or an FM-index of size $O(n)$ bits, one can support exact (0-mismatch) matching in $O(m + occ \log^\epsilon n)$ time, where occ is the number of occurrences and ε is any positive constant smaller than or equal to 1.*

For inexact matching ($k \neq 0$), there are solutions whose indexes can help answer a k-mismatch/k-difference pattern query for any $k \geq 0$. Those indexes are created by augmenting the suffix tree and its variants. Theorems 3 to 7 summarize the current best results in such direction.

Theorem 3 (Chan, Lam, Sung, Tam, and Wong, 2006 [3]) *Given an index of size $O(n)$ words, one can support k-mismatch lookup in $O(m + occ + (c \log n)^{k(k+1)} \cdot \log\log n)$ time where c is a positive constant. For k-difference lookup, the term occ becomes $k^3 3^k occ$.*

Theorem 4 (Chan, Lam, Sung, Tam, and Wong, 2006 [3]) *Given an index of size $O(n)$ bits, one can support k-mismatch lookup in $O(m + (occ + (c \log n)^{k(k+2)} \cdot \log\log n) \log^\epsilon n)$ time where c is a positive constant and ε is any positive constant smaller than or equal to 1. For k-difference lookup, the term occ becomes $k^3 3^k occ$.*

Theorem 5 (Lam, Sung, and Wong, 2005 [12]) *Given an index of size $O(n\sqrt{\log n})$ bits, one can support k-mismatch/k-difference lookup in $O((|\Sigma|m)^k(k + \log\log n) + occ)$ time.*

Theorem 6 (Lam, Sung, and Wong, 2005 [12]) *Given an index of size $O(n)$ bits, one can support k-mismatch/k-difference lookup in $O(\log^\epsilon((|\Sigma|m)^k(k + \log\log n) + occ))$ time where ε is any positive constant smaller than or equal to 1.*

Theorem 7 (Chan, Lam, Sung, Tam, and Wong, 2006 [4]) *Given an index of size $O(n)$ words, one can support k-mismatch lookup in $O(|\Sigma|^k m^{k-1} \log n \log\log n +$*

occ) time. For k-difference lookup, the term occ becomes $k^3 3^k occ$.

When k is given, one can create indexes whose sizes depend on k. Those solutions create the so-called k-error suffix tree and its variants. Theorems 8 to 11 summarize the current best results in this direction.

Theorem 8 (Maas and Nowak, 2005 [13]) *Given an index of size* $O(|\Sigma|^k n \log^k n)$ *words, one can support k-mismatch/k-difference lookup in* $O(m + occ)$ *expected time.*

Theorem 9 (Maas and Nowak, 2005 [13]) *Consider a uniformly and independently generated text of length n. One can construct an index of size* $O(|\Sigma|^k n \log^k n)$ *words on average, such that a k-mismatch/k-difference lookup query can be supported in* $O(m + occ)$ *worst case time.*

Theorem 10 (Chan, Lam, Sung, Tam, and Wong, 2006 [3]) *Given an index of size* $O(n \log^{k-h+1} n)$ *words where* $h \leq k$, *one can support k-mismatch lookup in* $O(m + occ + c^{k^2} \log^{\max\{kh, k+h\}} n \log\log n)$ *time where c is a positive constant. For k-difference lookup, the term occ becomes* $k^3 3^k occ$.

Theorem 11 (Chan, Lam, Sung, Tam, and Wong, 2006 [4]) *Given an index of size* $O(n \log^{k-1} n)$ *words, one can support k-mismatch lookup in* $O(m + occ + \log^k n \cdot \log\log n)$ *time. For k-difference lookup, the term occ becomes* $k^3 3^k occ$.

In addition, there are indexes which are efficient in practice for small k/m but give no worst case complexity guarantees. Those methods are based on filtration. The basic idea is to partition the pattern into short segments and locate those short segments in the text, allowing zero or a small number of errors. Those short segments help to identify candidate regions for the occurrences of the pattern. Finally, by verifying those candidate regions, one can recover all occurrences of the pattern. See [16] for a summary of those results. One of the best results based on filtration is stated in the following theorem.

Theorem 12 (Myers, 1994 [14]; Navarro and Baeza-Yates, 2000 [15]) *Consider an index of size* $O(n)$ *words. If* $k/m < 1 - O(1/\sqrt{\Sigma})$, *one can support a k-mismatch/k-difference search in* $O(n^\epsilon)$ *expected time where* ε *is a positive constant smaller than 1.*

All the above approaches either tried to index the strings with errors or are based on filtering. There are also solutions which use radically different approaches. For instance, there are solutions which transform approximate string searching into range queries in metric space [5].

Applications

Due to the advance in both internet and biological technologies, enormous text data is accumulated. For example, there is a 60G genomic sequence data in a gene bank. The data size is expected to grow exponentially.

To handle the huge data size, indexing techniques are vital to speed up the pattern matching queries. Moreover, exact pattern matching is no longer sufficient for both internet and biological data. For example, biological data usually contains a lot of differences due to experimental error and mutation and evolution. Therefore, approximate pattern matching becomes more appropriate. This gives the motivation for developing indexing techniques that allow pattern matching with errors.

Open Problems

The complexity for indexed approximate matching is still not fully understood. One would like to know the answers for a number of questions. For instance, one haves the following two questions. (1) Given a fixed index size of $O(n)$ words, what is the best time complexity of a k-mismatch/k-difference query? (2) If the k-mismatch/k-difference query time is fixed to $O(m + occ)$, what is the best space complexity of the index?

Cross References

- ▶ Text Indexing
- ▶ Two-Dimensional Pattern Indexing

Recommended Reading

1. Amir, A., Keselman, D., Landau, G.M., Lewenstein, M., Lewenstein, N., Rodeh, M.: Indexing and dictionary matching with one error. In: Proceedings of Workshop on Algorithms and Data Structures, 1999, pp. 181–192
2. Buchsbaum, A.L., Goodrich, M.T., Westbrook, J.R.: Range searching over tree cross products. In: Proceedings of European Symposium on Algorithms, 2000, pp. 120–131
3. Chan, H.-L., Lam, T.-W., Sung, W.-K., Tam, S.-L., Wong, S.-S.: A linear size index for approximate pattern matching. In: Proceedings of Symposium on Combinatorial Pattern Matching, 2006, pp. 49–59
4. Chan, H.-L., Lam, T.-W., Sung, W.-K., Tam, S.-L., Wong, S.-S.: Compressed indexes for approximate string matching. In: Proceedings of European Symposium on Algorithms, 2006, pp. 208–219
5. Navarro, G., Chávez, E.: A metric index for approximate string matching. Theor. Comput. Sci. **352**(1–3), 266–279 (2006)
6. Cobbs, A.: Fast approximate matching using suffix trees. In: Proceedings of Symposium on Combinatorial Pattern Matching, 1995, pp. 41–54
7. Coelho, L.P., Oliveira, A.L.: Dotted suffix trees: a structure for approximate text indexing. In: SPIRE, 2006, pp. 329–336

8. Cole, R., Gottlieb, L.A., Lewenstein, M.: Dictionary matching and indexing with errors and don't cares. In: Proceedings of Symposium on Theory of Computing, 2004, pp. 91–100
9. Ferragina, P., Manzini, G.: Opportunistic data structures with applications. In: Proceedings of Symposium on Foundations of Computer Science, 2000, pp. 390–398
10. Grossi, R., Vitter, J.S.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In: Proceedings of Symposium on Theory of Computing, 2000, pp. 397–406
11. Huynh, T.N.D., Hon, W.K., Lam, T.W., Sung, W.K.: Approximate string matching using compressed suffix arrays. In: Proceedings of Symposium on Combinatorial Pattern Matching, 2004, pp. 434–444
12. Lam, T.W., Sung, W.K., Wong, S.S.: Improved approximate string matching using compressed suffix data structures. In: Proceedings of International Symposium on Algorithms and Computation, 2005, pp. 339–348
13. Maaß, M.G., Nowak, J.: Text indexing with errors. In: Proceedings of Symposium on Combinatorial Pattern Matching, 2005, pp. 21–32
14. Myers, E.G.: A sublinear algorithm for approximate keyword searching. Algorithmica **12**, 345–374 (1994)
15. Navarro, G., Baeza-Yates R.: A hybrid indexing method for approximate string matching. J. Discret. Algorithms **1**(1), 205–209 (2000)
16. Navarro, G., Baeza-Yates, R.A., Sutinen, E., Tarhio, J.: Indexing methods for approximate string matching. IEEE Data Eng. Bull. **24**(4), 19–27 (2001)
17. Weiner., P.: Linear Pattern Matching Algorithms. In: Proceedings of Symposium on Switching and Automata Theory, 1973, pp. 1–11

Inductive Inference

1983; Case, Smith

Sandra Zilles
Department of Computing Science, University of Alberta, Edmonton, AB, Canada

Keywords and Synonyms

Induction; Learning from examples

Problem Definition

The theory of inductive inference is concerned with the capabilities and limitations of machine learning. Here the learning machine, the concepts to be learned, as well as the hypothesis space are modeled in recursion theoretic terms, based on the framework of identification in the limit [1,8].

Formally, considering recursive functions (mapping natural numbers to natural numbers) as target concepts, a learner (inductive inference machine) is supposed to process, step by step, gradually growing segments of the graph of a target function. In each step, the learner outputs a program in some fixed programming system, where successful learning means that the sequence of programs returned in this process eventually stabilizes on some program actually computing the target function.

Case and Smith [2,3] have proposed several variants of this model in order to study the influence that certain constraints or relaxations may have on the capabilities of learners, thereby restricting (i) the number of mind changes (i. e., changes of output programs) a learner is allowed for in this process and (ii) the number of errors the program eventually hypothesized may have when compared to the target function.

One major result of studying the corresponding effects is a hierarchy of inference types culminating in a model general enough to allow for the identification of the whole class of recursive functions by a single inductive inference machine.

Notations

The target concepts for learning in the model discussed below are recursive functions [13] mapping natural numbers to natural numbers. Such functions, as well as partial recursive functions in general, are considered as computable in an arbitrary, but fixed acceptable numbering $\varphi = (\varphi_i)_{i\in\mathbb{N}}$. Here $\mathbb{N} = \{0, 1, 2, \dots\}$ denotes the set of all natural numbers. ϕ is interpreted as a programming system, where each $i \in \mathbb{N}$ is called a program for the partial recursive function φ_i.

Suppose f and g are partial recursive functions and $n \in \mathbb{N}$. Below $f =^n g$ is written if the set $\{x \in \mathbb{N} \mid f(x) \neq g(x)\}$ is of cardinality at most n. If the set $\{x \in \mathbb{N} \mid f(x) \neq g(x)\}$ is finite, this is denoted by $f =^* g$. One considers $*$ as a special symbol for which the $<$-relation is extended by $n < *$ for all $n \in \mathbb{N}$. For any recursive f and any $z \in \mathbb{N}$, let $f[z]$ denote $(z, (f(0), \dots, f(z)))$ for short.

For further basic recursion theoretic notions, the reader is referred to [13].

Learning Models

Case and Smith [3] build their theory upon the fundamental model of identification in the limit [1,8]. There a learner can be understood as an algorithmic device, called an inductive inference machine, which, given any 'graph segment' $f[z]$ as its input, returns a program $i \in \mathbb{N}$. Such a learner M identifies a recursive function f in the limit, if there is some $j \in \mathbb{N}$ such that

$$\varphi_j = f \text{ and } M(f[z]) = j \text{ for all but finitely many } z \in \mathbb{N}.$$

A class of recursive functions is learnable in the limit, if there is an inductive inference machine identifying each function in the class in the limit. Identification in the limit is called EX-identification, since a program for f is termed an *ex*planation for f.

For instance, the class of all primitive recursive functions is EX-identifiable, whereas the class of all recursive functions is not [8].

The central questions discussed by Case and Smith [3] are how the limitations of EX-learners are affected by posing certain requirements on the success criterion, concerning

- convergence criteria,
  - e. g., when restricting the number of permitted mind changes,
  - e. g., when relaxing the constraints on syntactical convergence of the sequence of programs returned in the learning process,
- accuracy,
  - e. g., when relaxing the number of permitted anomalies in the programs returned eventually.

Problem 1 *In which way do modifications of* EX*-identification in terms of accuracy and convergence criteria affect the capabilities of the corresponding learners?*

Problem 2 *In particular, if inaccuracies are permitted, can* EX*-learners always refute inaccurate hypotheses?*

Problem 3 *How much relaxation of the model of* EX*-identification is needed to achieve learnability of the full class of recursive functions?*

Key Results

Accuracy and Convergence Constraints

In order to systematically address these problems, Case and Smith [3] have defined inference types reflecting restrictions and relaxations of EX-identification as follows.

Definition 1 Suppose S is a class of recursive functions and $m, n \in \mathbb{N} \cup \{*\}$. S is EX_n^m-identifiable, if there is an inductive inference machine M, such that for any function $f \in S$ there is some $j \in \mathbb{N}$ satisfying

- $M(f[z]) = j$ for all but finitely many $z \in \mathbb{N}$,
- $\varphi_j =^m f$, and
- the cardinality of the set $\{z \in \mathbb{N} \mid M(f[z]) \neq M(f[z+1])\}$ is at most n.

EX_n^m denotes the set of all classes of recursive functions which are EX_n^m-identifiable.

Definition 2 Suppose S is a class of recursive functions and $m \in \mathbb{N} \cup \{*\}$. S is BC^m-identifiable, if there is an inductive inference machine M, which, for any function $f \in S$, satisfies

- $\varphi_{M(f[z])} =^m f$ for all but finitely many $z \in \mathbb{N}$.

BC^m denotes the set of all classes of recursive functions which are BC^m-identifiable. BC is short for behaviorally correct—the difference to EX-learning is that convergence of the sequence of programs returned by the learner is defined only in terms of semantics, no longer in terms of syntax.

The Impact of Accuracy and Convergence Constraints

In general, each permission of mind changes or anomalies increases the capabilities of learners; however mind changes cannot be traded in for anomalies or vice versa.

Theorem 1 *Let $a, b, c, d \in \mathbb{N} \cup \{*\}$. Then $\mathrm{EX}_b^a \subseteq \mathrm{EX}_d^c$ if and only if $a \leq c$ and $b \leq d$.*

Corollary 1 *For any $m, n \in \mathbb{N}$ the following inclusions hold.*

1. $\mathrm{EX}_n^m \subset \mathrm{EX}_n^{m+1} \subset \mathrm{EX}_n^*$.
2. $\mathrm{EX}_n^m \subset \mathrm{EX}_{n+1}^m \subset \mathrm{EX}_*^m$.

Theorem 2 *Let $n \in \mathbb{N}$. Then $\mathrm{EX}_*^* \subset \mathrm{BC}^n \subset \mathrm{BC}^{n+1} \subset \mathrm{BC}^*$.*

These results are essential concerning Problem 1.

Refutability

In particular, refutability demands in the sense that every incorrect hypothesis should be refutable (see [12]) are not applicable in the theory of inductive inference, see Problem 2.

Formally, Case and Smith [3] consider refutability as a property guaranteed by Popperian machines, the latter being defined as follows:

Definition 3 Suppose M is an inductive inference machine M. M is Popperian if, on any input, M returns a program of a recursive function.

Results thereon include:

Theorem 3 *There is an* EX*-identifiable class S of recursive functions for which there is no Popperian IIM witnessing its* EX*-identifiability.*

Corollary 2 *There is an EX^1-identifiable class S of recursive functions for which there is no Popperian IIM witnessing its EX^1-identifiability.*

Additionally, in EX^1-identification, Popper's refutability principle can not be applied even if it concerns only those hypotheses returned in the limit.

Learning All Recursive Functions

Since the results above yield a hierarchy of inference types with strictly growing collections of learnable classes, there is also an implicit answer to Problem 3: the class of recursive functions is neither in EX_n^m for any $m, n \in \mathbb{N} \cup \{*\}$ nor in BC^m for any $m \in \mathbb{N}$. In contrast to that, Case and Smith [3] prove

Theorem 4 *The class of all recursive functions is in* BC^*.

Applications

The work of Case and Smith [3] has been of high impact in learning theory.

A consequence of the discussion of anomalies has been that refutability principles in general do not hold for identification in the limit. This result has given rise to later studies on methods and techniques inductive inference machines might apply in order to discover their errors [6] and thus to further insights into the nature of inductive inference.

Concerning the study of mind change hierarchies, among others, their lifting to transfinite ordinal numbers [7] is a notable extension.

Moreover, the theory of learning as proposed by Case and Smith [3] has been applied for the development of the theory of identifying recursive [10] or recursively enumerable [9] languages.

Open Problems

Among the currently open problems in inductive inference, one key challenge is to find a reasonable notion of the complexity of learning problems (i. e., of classes of recursive functions) involving the run-time complexity of learners as well as the number of mind changes required to learn the functions in a class. In particular, special natural classes of functions should be analyzed in terms of such a complexity notion.

Though of course the hierarchies $\mathrm{EX}_0^m \subset \mathrm{EX}_1^m \subset \mathrm{EX}_2^m \subset \ldots$ for any $m \in \mathbb{N}$ reflect some increase of complexity in that sense, a corresponding complexity notion would not address the aspect of run-time complexity of learners. Different complexity notions have been introduced, such as the so-called intrinsic complexity [5] (neglecting run-time complexity) and the 'measure under the curve' [4] (respecting the number of examples required, but neglecting the number of mind changes). In particular, for learning deterministic finite automata, different notions of run-time complexity have been discussed [11].

However, the definition of a more capacious complexity notion remains an open issue.

Cross References

► PAC Learning

Recommended Reading

1. Blum, L., Blum, M.: Toward a mathematical theory of inductive inference. Inform. Control **28**(2), 125–155 (1975)
2. Case, J., Smith, C.H.: Anomaly hierarchies of mechanized inductive inference. In: Proceedings of the 10th Symposium on the Theory of Computing, pp. 314–319. ACM, New York (1978)
3. Case, J., Smith, C.H.: Comparison of Identification Criteria for Machine Inductive Inference. Theor. Comput. Sci. **25**(2), 193–220 (1983)
4. Daley, R.P., Smith, C.H.: On the Complexity of Inductive Inference. Inform. Control **69**(1–3), 12–40 (1986)
5. Freivalds, R., Kinber, E., Smith, C.H.: On the Intrinsic Complexity of Learning. Inform. Comput. **118**(2), 208–226 (1995)
6. Freivalds, R., Kinber, E., Wiehagen, R.: How inductive inference strategies discover their errors. Inform. Comput. **123**(1), 64–71 (1995)
7. Freivalds, R., Smith, C.H.: On the Role of Procrastination in Machine Learning. Inform. Comput. **107**(2), 237–271 (1993)
8. Gold, E.M.: Language identification in the limit. Inform. Control **10**(5), 447–474 (1967)
9. Kinber, E.B., Stephan, F.: Language Learning from Texts: Mind-changes, Limited Memory, and Monotonicity. Inform. Comput. **123**(2), 224–241 (1995)
10. Lange, S., Grieser, G., Zeugmann, T.: Inductive inference of approximations for recursive concepts. Theor. Comput. Sci. **348**(1), 15–40 (2005)
11. Pitt, L.: Inductive inference, DFAs, and computational complexity. In: Analogical and Inductive Inference, 2nd International Workshop, Reinhardsbrunn Castle, GDR. Lecture Notes in Computer Science, vol. 397, pp. 18–44. Springer, Berlin (1989)
12. Popper, K.: The Logic of Scientific Discovery. Harper & Row, New York (1959)
13. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York (1967)

I/O-model

1988; Aggarwal, Vitter

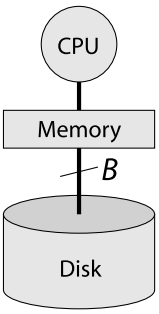
Norbert Zeh
Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada

Keywords and Synonyms

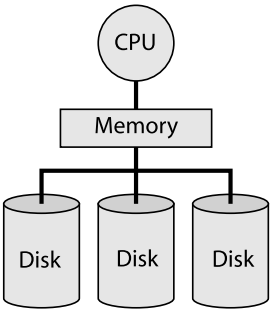
External-memory model; Disk access model (DAM)

Definition

The Input/Output model (I/O-model) [1] views the computer as consisting of a *processor*, *internal memory* (RAM), and *external memory* (disk). See Fig. 1. The internal mem-



I/O-model, Figure 1
The I/O-model



I/O-model, Figure 2
The parallel disk model

ory is of limited size, large enough to hold M data items. The external memory is of conceptually unlimited size and is divided into *blocks* of B consecutive data items. All computation has to happen on data in internal memory. Data is brought into internal memory and written back to external memory using *I/O-operations* (I/Os), which are performed explicitly by the algorithm. Each such operation reads or writes one block of data from or to external memory. The complexity of an algorithm in this model is the number of I/Os it performs.

The *parallel disk model* (PDM) [10] is an extension of the I/O-model that allows the external memory to consist of $D \geq 1$ parallel disks. See Fig. 2. In this model, a single I/O-operation is capable of reading or writing up to D independent blocks, as long as each of them is stored on a different disk.

Key Results

A few complexity bounds are of importance to virtually every I/O-efficient algorithm or data structure. The *searching bound* of $\Theta(\log_B n)$ I/Os, which can be achieved using a B-tree [4], is the cost of searching for an element in an ordered collection of n elements, using comparisons only. It is thus the equivalent of the $\Theta(\log n)$ searching bound in internal memory.

Scanning a list of n consecutive data items obviously takes $\lceil n/B \rceil$ I/Os or, in the PDM, $\lceil n/DB \rceil$ I/Os. This *scanning bound* is usually referred to as a "linear number of I/Os" because it is the equivalent of the $O(n)$ time bound required to do the same in internal memory.

The *sorting bound* of $\mathrm{sort}(n) = \Theta((n/B)\log_{M/B}(n/B))$ I/Os denotes the cost of sorting n elements using comparisons only. It is thus the equivalent of the $\Theta(n \log n)$ sorting bound in internal memory. In the PDM, the sorting bound becomes $\Theta((n/DB)\log_{M/B}(n/B))$. This bound can be achieved using a range of sorting algorithms, including external merge sort [1,6] and distribution sort [1,5].

Arguably the most interesting bound is the *permutation bound*, that is, the cost of rearranging n elements in a given order, which is $\Theta(\min(\mathrm{sort}(n), n))$ [1] or, in the PDM, $\Theta(\min(\mathrm{sort}(n), n/D))$ [10]. For all practical purposes, this is the same as the sorting bound. Note the contrast to internal memory where, up to constant factors, permuting has the same cost as a linear scan. Since almost all non-trivial algorithmic problems include a permutation problem, this implies that only exceptionally simple problems can be solved in $O(\mathrm{scan}(n))$ I/Os; most problems have an $\Omega(\mathrm{perm}(n))$, that is, essentially an $\Omega(\mathrm{sort}(n))$ lower bound. Therefore, while internal-memory algorithms aiming for linear time have to carefully avoid the use of sorting as a tool, external-memory algorithms can sort without fear of significantly exceeding the lower bound. This makes the design of I/O-optimal algorithms potentially easier than the design of optimal internal-memory algorithms. It is, however, counterbalanced by the fact that, unlike in internal memory, the sorting bound is *not* equal to n times the searching bound, which implies that algorithms based on querying a tree-based search structure $O(n)$ times usually do not translate into I/O-efficient algorithms. *Buffer trees* [3] achieve an amortized search bound of $O((1/B)\log_{M/B}(N/B))$ I/O, but can be used only if the entire update and query sequence is known in advance and thus provide only a limited solution to this problem.

Apart from these fundamental results, there exist a wide range of interesting techniques, particularly for solving geometric and graph problems. For surveys, refer to [2,9].

Applications

Modern computers are equipped with memory hierarchies consisting of several levels of cache memory, main mem-

ory (RAM), and disk(s). Access latencies increase with the distance from the processor, as do the sizes of the memory levels. To amortize these increasing access latencies, data are transferred between different levels of cache in blocks of consecutive data items. As a result, the cost of a memory access depends on the level in the memory hierarchy currently holding the data item—the difference in access latency between L1 cache and disk is about 10^6—and the cost of a sequence of accesses to data items stored at the same level depends on the number of blocks over which these items are distributed.

Traditionally, algorithms have been designed to minimize the number of computation steps; the access locality necessary to solve a problem using few data transfers between memory levels has been largely ignored. Hence, the designed algorithms work well on data sets of moderate size, but do not take noticeable advantage of cache memory and usually break down completely in out-of-core computations. Since the difference in access latencies is largest between main memory and disk, the I/O-model focuses on minimizing this I/O-bottleneck. This two-level view of the memory hierarchy keeps the model simple and useful for analyzing sophisticated algorithms, while providing a good prediction of their practical performance.

Much effort has been made already to translate provably I/O-efficient algorithms into highly efficient implementations. Examples include TPIE [8] and STXXL [7], two libraries that aim to provide highly optimized and powerful primitives for the implementation of I/O-efficient algorithms. In spite of these efforts, a significant gap between the theory and practice of I/O-efficient algorithms remains (see next section).

Open Problems

There are a substantial number of open problems in the area of I/O-efficient algorithms. The most important ones concern graph and geometric problems.

Traditional graph algorithms usually use a well-organized graph traversal such as depth-first search or breadth-first search to gain information about the structure of the graph and then use this information to solve the problem at hand. In the I/O-model, no I/O-efficient depth-first search algorithm is known and for breadth-first search and shortest paths, progress has been made only recently on undirected graphs. For directed graphs, even such simple problems as deciding whether there exists a directed path between two vertices are currently open. The main research focus in this area is therefore to either develop I/O-efficient general traversal algorithms or to continue the current strategy of devising graph algorithms that depart from traditional traversal-based approaches.

Techniques for solving geometric problems I/O-efficiently are much better understood than is the case for graph algorithms, at least in two dimensions. Nevertheless, there are a few important frontiers that remain. Arguably the most important one is the development of I/O-efficient algorithms and data structures for higher-dimensional geometric problems. Motivated by database applications, higher-dimensional range searching is one of the problems to be studied in this context. Little work has been done in the past on solving proximity problems, which pose another frontier currently explored by researchers in the field. Motivated by the need for such structures in a range of application areas and in particular in geographic information systems, there has been some recent focus on the development of multifunctional data structures, that is, structures that can answer different types of queries efficiently. Most existing structures are carefully tuned to efficiently support *one* particular type of query.

For both, I/O-efficient graph algorithms and computational geometry, there is a substantial gap between the obtained theoretical results and what is known to be practical, even though more experimental work has been done on geometric algorithms than on graph algorithms. Thus, if I/O-efficient algorithms in these areas are to have any practical impact, increased efforts are needed to bridge this gap by developing practically I/O-efficient algorithms that are still *provably* efficient.

Cross References

For details on ► External Sorting and Permuting, please refer to the corresponding entry. Details on one- and higher-dimensional searching are provided in the entries on ► B-trees and ► R-trees. The reader interested in algorithms that focus on efficiency at all levels of the memory hierarchy should consult the entry on the ► Cache-Oblivious Model.

Recommended Reading

1. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Commun. ACM **31**(9), 1116–1127 (1988)
2. Arge, L.: External memory data structures. In: Abello, J., Pardalos, P.M., Resende, M.G.C. (eds.) Handbook of Massive Data Sets, pp. 313–357. Kluwer Academic Publishers, Dordrecht (2002)
3. Arge, L.: The buffer tree: A technique for designing batched external data structures. Algorithmica **37**(1), 1–24 (2003)

4. Bayer, R., McCreight, E.: Organization of large ordered indexes. Acta Inform. **1**, 173–189 (1972)
5. Nodine, M.H., Vitter, J.S.: Deterministic distribution sort in shared and distributed memory multiprocessors. In: Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 120–129. Velen, June/July 1993
6. Nodine, M.H., Vitter, J.S.: Greed Sort: An optimal sorting algorithm for multiple disks. J. ACM **42**(4), 919–933 (1995)
7. STXXL: C++ Standard Library for Extra Large Data Sets. http://stxxl.sourceforge.net. Accessed: 15 March 2008
8. TPIE — A Transparent Parallel I/O-Environment. http://www.cs.duke.edu/TPIE. Accessed: 15 March 2008
9. Vitter, J.S.: External memory algorithms and data structures: Dealing with massive data. ACM Comput. Surv. **33**(2), 209–271 (2001)
10. Vitter, J.S., Shriver, E.A.M.: Algorithms for parallel memory I: Two-level memories. Algorithmica **12**(2–3), 110–147 (1994)