A

Abelian Hidden Subgroup Problem

1995; Kitaev

Michele Mosca[1,2]
[1] Combinatorics and Optimization / Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada
[2] Perimeter Institute for Theoretical Physics, St. Jerome's University, Waterloo, ON, Canada

Keywords and Synonyms

Generalization of Abelian stabilizer problem; Generalization of Simon's problem

Problem Definition

The Abelian hidden subgroup problem is the problem of finding generators for a subgroup K of an Abelian group G, where this subgroup is defined implicitly by a function $f\colon G \to X$, for some finite set X. In particular, f has the property that $f(v) = f(w)$ if and only if the cosets[1] $v + K$ and $w + K$ are equal. In other words, f is constant on the cosets of the subgroup K, and distinct on each coset.

It is assumed that the group G is finitely generated and that the elements of G and X have unique binary encodings (the binary assumption is not so important, but it is important to have unique encodings.) When using variables g and h (possibly with subscripts) multiplicative notation is used for the group operations. Variables x and y (possibly with subscripts) will denote integers with addition. The boldface versions $\mathbf{x}$ and $\mathbf{y}$ will denote *tuples* of integers or binary strings.

By assumption, there is computational means of computing the function f, typically a circuit or "black box" that maps the encoding of a value g to the encoding of $f(g)$. The theory of reversible computation implies that one can turn a circuit for computing $f(g)$ into a reversible circuit for computing $f(g)$ with a modest increase in the size of the circuit. Thus it will be assumed that there is a reversible circuit or black box that maps $(g, \mathbf{z}) \mapsto (g, \mathbf{z} \oplus f(g))$, where $\oplus$ denotes the bitwise *XOR* (sum modulo 2), and $\mathbf{z}$ is any binary string of the same length as the encoding of $f(g)$.

Quantum mechanics implies that any reversible gate can be extended linearly to a unitary operation that can be implemented in the model of quantum computation. Thus, it is assumed that there is a quantum circuit or black box that implements the unitary map $U_f\colon |g\rangle|\mathbf{z}\rangle \mapsto |g\rangle|\mathbf{z} \oplus f(g)\rangle$.

Although special cases of this problem have been considered in classical computer science, the general formulation as the hidden subgroup problem seems to have appeared in the context of quantum computing, since it neatly encapsulates a family of "black-box" problems for which quantum algorithms offer an exponential speed up (in terms of query complexity) over classical algorithms. For some explicit problems (i. e., where the black box is replaced with a specific function, such as exponentiation modulo N), there is a conjectured exponential speed up.

Abelian Hidden Subgroup Problem

Input: Elements $g_1, g_2, \ldots, g_n \in G$ that generate the Abelian group G. A black box that implements $U_f\colon |m_1, m_2, \ldots, m_n\rangle|\mathbf{y}\rangle \mapsto |m_1, m_2, \ldots, m_n\rangle|f(g) \oplus \mathbf{y}\rangle$, where $g = g_1^{m_1} g_2^{m_2} \ldots g_n^{m_n}$, and K is the hidden subgroup corresponding to f.
Output: Elements $h_1, h_2, \ldots, h_l \in G$ that generate K.

Here we use multiplicative notation for the group G in order to be consistent with Kitaev's formulation of the Abelian stabilizer problem. Many of the applications of interest typically use additive notation for the group G.

It is hard to trace the precise origin of this general formulation of the problem, which simultaneously general-

[1] Assuming additive notation for the group operation here.

izes “Simon’s problem” [16], the order-finding problem (which is the quantum part of the quantum factoring algorithm [14]) and the discrete logarithm problem.

One of the earliest generalizations of Simon’s problem, the order-finding problem, and the discrete logarithm problem, which captures the essence of the Abelian hidden subgroup problem is the *Abelian stabilizer problem*, which was solved by Kitaev [11] using a quantum algorithm in his 1995 paper (and the solution also appears in [12]).

Let G be a group acting on a finite set X. That is, each element of G acts as a map from X to X in such a way that for any two elements $g, h \in G$, $g(h(z)) = (gh)(z)$ for all $z \in X$. For a particular element $z \in X$, the set of elements that fix z (that is the elements $g \in G$ such that $g(z) = z$) form a subgroup. This subgroup is called the stabilizer of z in G, denoted $St_G(z)$.

Abelian Stabilizer Problem

Input: Elements $g_1, g_2, \ldots, g_n \in G$ that generate the group G. An element $z \in X$. A black box that implements $U_{(G,X)} : |m_1, m_2, \ldots, m_n\rangle|z\rangle \mapsto |m_1, m_2, \ldots, m_n\rangle|g(z)\rangle$ where $g = g_1^{m_1} g_2^{m_2} \ldots g_n^{m_n}$.
Output: Elements $h_1, h_2, \ldots, h_l \in G$ that generate $St_G(z)$.

Let f_z denote the function from G to X that maps $g \in G$ to $g(z)$. One can implement U_{f_z} using $U_{(G,X)}$. The hidden subgroup corresponding to f_z is $St_G(z)$. Thus, the Abelian stabilizer problem is a special case of the Abelian hidden subgroup problem.

One of the subtle differences (discussed in Appendix 6 of [10]) between the above formulation of the Abelian stabilizer problem and the Abelian hidden subgroup problem is that Kitaev’s formulation gives a black box that for any $g, h \in G$ maps $|m_1, \ldots, m_n\rangle|f_z(g)\rangle \mapsto |m_1, \ldots, m_n\rangle|f_z(hg)\rangle$, where $g = g_1^{m_1} g_2^{m_2} \ldots g_n^{m_n}$ and estimates eigenvalues of shift operations of the form $|f_z(g)\rangle \mapsto |f_z(hg)\rangle$. In general, these shift operators are not explicitly needed, and it suffices to be able to compute a map of the form $|\mathbf{y}\rangle \mapsto |f_z(h) \oplus \mathbf{y}\rangle$ for any binary string $\mathbf{y}$.

Generalizations of this form have been known since shortly after Shor presented his factoring and discrete logarithm algorithms. For example, in [18] the hidden subgroup problem was discussed for a large class of finite Abelian groups, and more generally in [2] for any finite Abelian group presented as a product of finite cyclic groups. In [13] the Abelian hidden subgroup algorithm was related to eigenvalue estimation.

Other problems which can be formulated in this way include the following.

Deutsch’s Problem

Input: A black box that implements $U_f : |x\rangle|b\rangle \mapsto |x\rangle|b \oplus f(x)\rangle$, for some function f that maps $\mathbb{Z}_2 = \{0, 1\}$ to $\{0, 1\}$.
Output: “Constant” if $f(0) = f(1)$, “balanced” if $f(0) \neq f(1)$.

Note that $f(x) = f(y)$ if and only if $x - y \in K$, where K is either $\{0\}$ or $\mathbb{Z}_2 = \{0, 1\}$. If $K = \{0\}$ then f is $1-1$ or “balanced” and if $K = \mathbb{Z}_2$ then f is constant [4,5].

Simon’s Problem

Input: A black box that implements $U_f : |\mathbf{x}\rangle|\mathbf{b}\rangle \mapsto |\mathbf{x}\rangle|\mathbf{b} \oplus f(\mathbf{x})\rangle$ for some function f from $\mathbb{Z}_2^n$ to some set X (which is assumed to consist of binary strings of some fixed length) with the property that $f(\mathbf{x}) = f(\mathbf{y})$ if and only if $\mathbf{x} - \mathbf{y} \in K = \{\mathbf{0}, \mathbf{s}\}$ for some $\mathbf{s} \in \mathbb{Z}_2^n$.
Output: The “hidden” string $\mathbf{s}$.

The decision version allows $K = \{\mathbf{0}\}$ and asks whether K is trivial. Simon [16] presented an efficient algorithm for solving this problem, and an exponential lower bound on the query complexity. The solution to the Abelian hidden subgroup problem is a generalization of Simon’s algorithm (which deals with finite groups with many generators) and Shor’s algorithms [14,12] (which deal with an infinite group with one generator, and a finite group with two generators).

Key Results

Theorem (Abelian stabilizer problem) *There exists a quantum algorithm that, given an instance of the Abelian stabilizer problem, makes $n + O(1)$ queries to $U_{(G,X)}$, uses poly(n) other elementary quantum and classical operations, and with probability at least 2/3 outputs values $h_1, h_2, \ldots, h_l$ such that $St_G(z) = \langle h_1\rangle \oplus \langle h_2\rangle \oplus \cdots \langle h_l\rangle$.*

Kitaev first solved this problem (with a slightly higher query complexity, because his eigenvalue estimation procedure was not optimal). An eigenvalue estimation procedure based on the quantum Fourier transform achieves the $n + O(1)$ query complexity.

Theorem (Abelian hidden subgroup problem) *There exists a quantum algorithm that, given an instance of the Abelian hidden subgroup problem, makes $n + O(1)$ queries to U_f, uses poly(n) other elementary quantum and classical operations, and with probability at least 2/3 outputs values $h_1, h_2, \ldots, h_l$ such that $K = \langle h_1\rangle \oplus \langle h_2\rangle \oplus \cdots \langle h_l\rangle$.*

In some cases, the success probability can be made 1 with the same complexity, and in general the success probability can be made $1 - \epsilon$ using $n + O(\log(1/\epsilon))$ queries and

$poly(n, \log(1/\epsilon))$ other elementary quantum and classical operations.

Applications

Most of these applications in fact were known before the Abelian stabilizer problem or the Abelian hidden subgroup problem were formulated.

Finding the Order of an Element in a Group Let a be an element of a group H (which does *not* need to be Abelian). Consider the function f from $G = \mathbb{Z}$ to the group H where $f(x) = a^x$ for some element a of H. Then $f(x) = f(y)$ if and only if $x - y \in r\mathbb{Z}$. The hidden subgroup is $K = r\mathbb{Z}$ and a generator for K gives the order r of a [14,12].

Discrete Logarithms Let a be an element of a group H (which does *not* need to be Abelian), with $a^r = 1$, and suppose $b = a^k$ from some unknown k. The integer k is called the *discrete logarithm of b to the base a*. Consider the function f from $G = \mathbb{Z}_r \times \mathbb{Z}_r$ to H satisfying $f(x_1, x_2) = a^{x_1} b^{x_2}$. Then $f(x_1, x_2) = f(y_1, y_2)$ if and only if $(x_1, x_2) - (y_1, y_2) \in \{(tk, -t), t = 0, 1, \dots, r-1\}$, which is the subgroup $\langle(k, -1)\rangle$ of $\mathbb{Z}_r \times \mathbb{Z}_r$. Thus, finding a generator for the hidden subgroup K will give the discrete logarithm k. Note that this algorithm works for H equal to the multiplicative group of a finite field, or the additive group of points on an elliptic curve, which are groups that are used in public-key cryptography.

Hidden Linear Functions Let σ be some permutation of $\mathbb{Z}_N$ for some integer N. Let h be a function from $G = \mathbb{Z} \times \mathbb{Z}$ to $\mathbb{Z}_N$, $h(x, y) = x + ay \mod N$. Let $f = \sigma \circ h$. The hidden subgroup of f is $\langle(-a, 1)\rangle$. Boneh and Lipton [1] showed that even if the linear structure of h is hidden (by σ), one can efficiently recover the parameter a with a quantum algorithm.

Self-shift-equivalent Polynomials Given a polynomial P in l variables $X_1, X_2, \dots, X_l$ over $\mathbb{F}_q$, the function f that maps $(a_1, a_2, \dots, a_l) \in \mathbb{F}_q^l$ to $P(X_1 - a_1, X_2 - a_2, \dots, X_l - a_l)$ is constant on cosets of a subgroup K of $\mathbb{F}_q^l$. This subgroup K is the set of shift-self-equivalences of the polynomial P. Grigoriev [8] showed how to compute this subgroup.

Decomposition of a Finitely Generated Group Let G be a group with a unique binary representation for each element of G, and assume that the group operation, and recognizing if a binary string represents an element of G or not, can be done efficiently.

Given a set of generators $g_1, g_2, \dots, g_n$ for a group G, output a set of elements $h_1, h_2, \dots, h_l, l \leq n$, from the group G such that $G = \langle g_1\rangle \oplus \langle g_2\rangle \oplus \dots \oplus \langle g_l\rangle$. Such a generating set can be found efficiently [3] from generators of the hidden subgroup of the function that maps $(m_1, m_2, \dots, m_n) \mapsto g_1^{m_1} g_2^{m_2} \dots g_n^{m_n}$.

Discussion: What About non-Abelian Groups?

The great success of quantum algorithms for solving the Abelian hidden subgroup problem leads to the natural question of whether it can solve the hidden subgroup problem for non-Abelian groups. It has been shown that a polynomial number of queries suffice [7]; however, in general there is no bound on the overall computational complexity (which includes other elementary quantum or classical operations).

This question has been studied by many researchers, and efficient quantum algorithms can be found for some non-Abelian groups. However, at present, there is no efficient algorithm for most non-Abelian groups. For example, solving the hidden subgroup problem for the symmetric group would directly solve the graph automorphism problem.

Cross References

▶ Graph Isomorphism
▶ Quantum Algorithm for the Discrete Logarithm Problem
▶ Quantum Algorithm for Factoring
▶ Quantum Algorithm for the Parity Problem
▶ Quantum Algorithm for Solving the Pell's Equation

Recommended Reading

1. Boneh, D., Lipton, R.: Quantum Cryptanalysis of Hidden Linear Functions (Extended Abstract) In: Proceedings of 15th Annual International Cryptology Conference (CRYPTO'95), pp. 424–437, Santa Barbara, 27–31 August 1995
2. Brassard, G., Høyer, P.: An exact quantum polynomial-time algorithm for Simon's problem. In: Proc. of Fifth Israeli Symposium on Theory of Computing ans Systems (ISTCS'97), pp. 12–23 (1997) and in: Proceedings IEEE Computer Society, Ramat-Gan, 17–19 June 1997
3. Cheung, K., Mosca, M.: Decomposing Finite Abelian Groups. Quantum Inf. Comp. **1**(2), 26–32 (2001)
4. Cleve, R., Ekert, A., Macchiavello, C., Mosca, M.: Quantum Algorithms Revisited. Proc. Royal Soc. London A **454**, 339–354 (1998)
5. Deutsch, D.: Quantum theory, the Church-Turing principle and the universal quantum computer. Proc. Royal Soc. London A **400**, 97–117 (1985)
6. Deutsch, D., Jozsa, R.: Rapid solutions of problems by quantum computation. Proc. Royal Soc. London A **439**, 553–558 (1992)

7. Ettinger, M., Høyer, P., Knill, E.: The quantum query complexity of the hidden subgroup problem is polynomial. Inf. Process. Lett. **91**, 43–48 (2004)
8. Grigoriev, D.: Testing Shift-Equivalence of Polynomials by Deterministic, Probabilistic and Quantum Machines. Theor. Comput. Sci. **180**, 217–228 (1997)
9. Høyer, P.: Conjugated operators in quantum algorithms. Phys. Rev. A **59**(5), 3280–3289 (1999)
10. Kaye, P., Laflamme, R., Mosca, M.: An Introduction to Quantum Computation. Oxford University Press, Oxford (2007)
11. Kitaev, A.: Quantum measurements and the Abelian Stabilizer Problem. quant-ph/9511026, http://arxiv.org/abs/quant-ph/9511026 (1995) and in: Electronic Colloquium on Computational Complexity (ECCC) 3, Report TR96-003,http://eccc.hpi-web.de/eccc-reports/1995/TR96-003/ (1996)
12. Kitaev, A.Y.: Quantum computations: algorithms and error correction. Russ. Math. Surv. **52**(6), 1191–1249 (1997)
13. Mosca, M., Ekert, A.: The Hidden Subgroup Problem and Eigenvalue Estimation on a Quantum Computer. In: Proceedings 1st NASA International Conference on Quantum Computing & Quantum Communications. Lecture Notes in Computer Science, vol. 1509, pp. 174–188. Springer, London (1998)
14. Shor, P.: Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pp. 124–134, Santa Fe, 20–22 November 1994
15. Shor, P.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM J. Comp. **26**, 1484–1509 (1997)
16. Simon, D.: On the power of quantum computation. In: Proceedings of the 35th IEEE Symposium on the Foundations of Computer Science (FOCS), pp. 116–123, Santa Fe, 20–22 November 1994
17. Simon, D.: On the Power of Quantum Computation. SIAM J. Comp. **26**, 1474–1483 (1997)
18. Vazirani, U.: Berkeley Lecture Notes. Fall 1997. Lecture 8. http://www.cs.berkeley.edu/~vazirani/qc.html (1997)

Adaptive Partitions

1986; Du, Pan, Shing

PING DENG[1], WEILI WU[1], EUGENE SHRAGOWITZ[2]

[1] Department of Computer Science,
University of Texas at Dallas, Richardson, TX, USA

[2] Department of Computer Science and Engineering,
University of Minnesota, Minneapolis, MN, USA

Keywords and Synonyms

Technique for constructing approximation

Problem Definition

Adaptive partition is one of major techniques to design polynomial-time approximation algorithms, especially polynomial-time approximation schemes for geometric optimization problems. The framework of this technique is to put the input data into a rectangle and partition this rectangle into smaller rectangles by a sequence of cuts so that the problem is also partitioned into smaller ones. Associated with each adaptive partition, a feasible solution can be constructed recursively from solutions in smallest rectangles to bigger rectangles. With dynamic programming, an optimal adaptive partition is computed in polynomial time.

Historical Background

The adaptive partition was first introduced to the design of an approximation algorithm by Du et al. [5] with a guillotine cut while they studied the minimum edge length rectangular partition (MELRP) problem. They found that if the partition is performed by a sequence of guillotine cuts, then an optimal solution can be computed in polynomial time with dynamic programming. Moreover, this optimal solution can be used as a pretty good approximation solution for the original rectangular partition problem. Both Arora [1] and Mitchell et al. [12,13] found that the cut needs not to be completely guillotine. In other words, the dynamic programming can still runs in polynomial time if subproblems have some relations but the number of relations is smaller. As the number of relations goes up, the approximation solution obtained approaches the optimal one, while the run time, of course, goes up. They also found that this technique can be applied to many geometric optimization problems to obtain polynomial-time approximation schemes.

Key Results

The MELRP was proposed by Lingas et al. [9] as follows: Given a rectilinear polygon possibly with some rectangular holes, partition it into rectangles with minimum total edge length. Each hole may be degenerated into a line segment or a point.

There are several applications mentioned in [9] for the background of the problem: process control (stock cutting), automatic layout systems for integrated circuit (channel definition), and architecture (internal partitioning into offices). The *minimum edge length* partition is a natural goal for these problems since there is a certain amount of waste (e. g., sawdust) or expense incurred (e. g., for dividing walls in the office) which is proportional to the sum of edge lengths drawn. For very large scale integration (VLSI) design, this criterion is used in the MIT Placement and Interconnect (PI) System to divide the routing region up into channels - one finds that this produces large “natural-looking” channels with a minimum of channel-to-channel interaction to consider.

They showed that while the MELRP in general is nondeterministic polynomial-time (NP) hard, is can be solved in time $O(n^4)$ in the hole-free case, where n is the number of vertices in the input rectilinear polygon. The polynomial algorithm is essentially a dynamic programming based on the fact that there always exists an optimal solution satisfying the property that every cut line passes through a vertex of the input polygon or holes (namely, every maximal cut segment is incident to a vertex of input or holes).

A naive idea to design an approximation algorithm for the general case is to use a forest connecting all holes to the boundary and then to solve the resulting hole-free case in $O(n^4)$ time. With this idea, Lingas [10] gave the first constant-bounded approximation; its performance ratio is 41.

Motivated by a work of Du et al. [4] on application of dynamic programming to optimal routing trees, Du et al. [5] initiated an idea of adaptive partition. They used a sequence of guillotine cuts to do rectangular partition; each guillotine cut breaks a connected area into at least two parts. With dynamic programming, they were able to show that a minimum-length guillotine rectangular partition (i. e., one with minimum total length among all guillotine partitions) can be computed in $O(n^5)$ time. Therefore, they suggested using the minimum-length guillotine rectangular partition to approximate the MELRP and tried to analyze the performance ratio. Unfortunately, they failed to get a constant ratio in general and only obtained a upper bound of 2 for the performance ratio in a NP-hard special case [7]. In this special case, the input is a rectangle with some points inside. Those points are holes. The following is a simple version of the proof obtained by Du et al. [6].

Theorem *The minimum-length guillotine rectangular partition is an approximation with performance ratio 2 for the MELRP.*

Proof Consider a rectangular partition P. Let $proj_x(P)$ denote the total length of segments on a horizontal line covered by vertical projection of the partition P.

A rectangular partition is said to be covered by a guillotine partition if each segment in the rectangular partition is covered by a guillotine cut of the latter. Let $guil(P)$ denote the minimum length of the guillotine partition covering P and $length(P)$ denote the total length of rectangular partition P. It will be proved by induction on the number k of segments in P that

$$guil(P) \le 2 \cdot length(P) - proj_x(P) .$$

For $k = 1$, one has $guil(P) = length(P)$. If the segment is horizontal, then one has $proj_x(P) = length(P)$ and hence

$$guil(P) = 2 \cdot length(P) - proj_x(P) .$$

If the segment is vertical, then $proj_x(P) = 0$ and hence

$$guil(P) < 2 \cdot length(P) - proj_x(P) .$$

Now, consider $k \ge 2$. Suppose that the initial rectangle has each vertical edge of length a and each horizontal edge of length b. Consider two cases:

Case 1. There exists a vertical segment s having length greater than or equal to $0.5a$. Apply a guillotine cut along this segment s. Then the remainder of P is divided into two parts P_1 and P_2 which form rectangular partition of two resulting small rectangles, respectively. By induction hypothesis,

$$guil(P_i) \le 2 \cdot length(P_i) - proj_x(P_i)$$

for $i = 1, 2$. Note that

$$\begin{aligned} guil(P) &\le guil(P_1) + guil(P_2) + a , \\ length(P) &= length(P_1) + length(P_2) + length(s) , \\ proj_x(P) &= proj_x(P_1) + proj_x(P_2) . \end{aligned}$$

Therefore,

$$guil(P) \le 2 \cdot length(P) - proj_x(P) .$$

Case 2. No vertical segment in P has length greater than or equal to $0.5a$. Choose a horizontal guillotine cut which partitions the rectangle into two equal parts. Let P_1 and P_2 denote rectangle partitions of the two parts, obtained from P. By induction hypothesis,

$$guil(P_i) \le 2 \cdot length(P_i) - proj_x(P_i)$$

for $i = 1, 2$. Note that

$$\begin{aligned} guil(P) &= guil(P_1) + guil(P_2) + b , \\ length(P) &\ge length(P_1) + length(P_2) , \\ proj_x(P) &= proj_x(P_1) = proj_x(P_2) = b . \end{aligned}$$

Therefore,

$$guil(P) \le 2 \cdot length(P) - proj_x(P) .$$

Gonzalez and Zheng [8] improved this upper bound to 1.75 and conjectured that the performance ratio in this case is 1.5.

Applications

In 1996, Arora [1] and Mitchell et al. [12,13,14] found that the cut does not necessarily have to be completely guillotine in order to have a polynomial-time computable optimal solution for such a sequence of cuts. Of course, the number of connections left by an incomplete guillotine cut should be limited. While Mitchell et al. developed the m-guillotine subdivision technique, Arora employed a "portal" technique. They also found that their techniques can be used for not only the MELRP, but also for many geometric optimization problems [1,2,3,12,13,14,15].

Open Problems

One current important submicron step of technology evolution in electronics interconnects has become the dominating factor in determining VLSI performance and reliability. Historically a problem of interconnects design in VLSI has been very tightly intertwined with the classical problem in computational geometry: Steiner minimum tree generation. Some essential characteristics of VLSI are roughly proportional to the length of the interconnects. Such characteristics include chip area, yield, power consumption, reliability and timing. For example, the area occupied by interconnects is proportional to their combined length and directly impacts the chip size. Larger chip size results in reduction of yield and increase in manufacturing cost. The costs of other components required for manufacturing also increase with increase of the wire length. From the performance angle, longer interconnects cause an increase in power dissipation, degradation of timing and other undesirable consequences. That is why finding the minimum length of interconnects consistent with other goals and constraints is such an important problem at this stage of VLSI technology.

The combined length of the interconnects on a chip is the sum of the lengths of individual signal nets. Each signal net is a set of electrically connected terminals, where one terminal acts as a driver and other terminals are receivers of electrical signals. Historically, for the purpose of finding an optimal configuration of interconnects, terminals were considered as points on the plane, and a routing problem for individual nets was formulated as a classical Steiner minimum tree problem. For a variety of reasons VLSI technology implements only rectilinear wiring on the set of parallel planes, and, consequently, with few exceptions, only a rectilinear version of the Steiner tree is being considered in the VLSI domain. This problem is known as the RSMT.

Further progress in VLSI technology resulted in more factors than just length of interconnects gaining importance in selection of routing topologies. For example, the presence of obstacles led to reexamination of techniques used in studies of the rectilinear Steiner tree, since many classical techniques do not work in this new environment. To clarify the statement made above, we will consider the construction of a rectilinear Steiner minimum tree in the presence of obstacles.

Let us start with a rectilinear plane with obstacles defined as rectilinear polygons. Given n points on the plane, the objective is to find the shortest rectilinear Steiner tree that interconnects them. One already knows that a polynomial-time approximation scheme for RSMT without obstacles exists and can be constructed by adaptive partition with application of either the portal or the m-guillotine subdivision technique. However, both the m-guillotine cut and the portal techniques do not work in the case that obstacles exists. The portal technique is not applicable because obstacles may block movement of the line that crosses the cut at a portal. The m-guillotine cut could not be constructed either, because obstacles may break down the cut segment that makes the Steiner tree connected.

In spite of the facts stated above, the RSMT with obstacles may still have polynomial-time approximation schemes.Strong evidence was given by Min et al. [11]. They constructed a polynomial-time approximation scheme for the problem with obstacles under the condition that the ratio of the longest edge and the shortest edge of the minimum spanning tree is bounded by a constant. This design is based on the classical nonadaptive partition approach. All of the above make us believe that a new adaptive technique can be found for the case with obstacles.

Cross References

▶ Metric TSP
▶ Rectilinear Steiner Tree
▶ Steiner Trees

Recommended Reading

1. Arora, S.: Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. In: Proc. 37th IEEE Symp. on Foundations of Computer Science, 1996, pp. 2–12
2. Arora, S.: Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In: Proc. 38th IEEE Symp. on Foundations of Computer Science, 1997, pp. 554–563
3. Arora, S.: Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. J. ACM **45**, 753–782 (1998)
4. Du, D.Z., Hwang, F.K., Shing, M.T., Witbold, T.: Optimal routing trees. IEEE Trans. Circuits **35**, 1335–1337 (1988)
5. Du, D.-Z., Pan, L.-Q., Shing, M.-T.: Minimum edge length guillotine rectangular partition. Technical Report 0241886, Math. Sci. Res. Inst., Univ. California, Berkeley (1986)
6. Du, D.-Z., Hsu, D.F., Xu, K.-J.: Bounds on guillotine ratio. Congressus Numerantium **58**, 313–318 (1987)

7. Gonzalez, T., Zheng, S.Q.: Bounds for partitioning rectilinear polygons. In: Proc. 1st Symp. on Computational Geometry (1985)
8. Gonzalez, T., Zheng, S.Q.: Improved bounds for rectangular and guillotine partitions. J. Symb. Comput. **7**, 591–610 (1989)
9. Lingas, A., Pinter, R.Y., Rivest, R.L., Shamir, A.: Minimum edge length partitioning of rectilinear polygons. In: Proc. 20th Allerton Conf. on Comm. Control and Compt., Illinos (1982)
10. Lingas, A.: Heuristics for minimum edge length rectangular partitions of rectilinear figures. In: Proc. 6th GI-Conference, Dortmund, January 1983. Springer
11. Min, M., Huang, S.C.-H., Liu, J., Shragowitz, E., Wu, W., Zhao, Y., Zhao, Y.: An Approximation Scheme for the Rectilinear Steiner Minimum Tree in Presence of Obstructions. Fields Inst. Commun. **37**, 155–164 (2003)
12. Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric *k*-MST problem. In: Proc. 7th ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 402–408.
13. Mitchell, J.S.B., Blum, A., Chalasani, P., Vempala, S.: A constant-factor approximation algorithm for the geometric *k*-MST problem in the plane. SIAM J. Comput. **28**(3), 771–781 (1999)
14. Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: Part II – A simple polynomial-time approximation scheme for geometric *k*-MST, TSP, and related problem. SIAM J. Comput. **29**(2), 515–544 (1999)
15. Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: Part III – Faster polynomial-time approximation scheme for geometric network optimization, manuscript, State University of New York, Stony Brook (1997)

Ad-Hoc Networks

▶ Channel Assignment and Routing in Multi-Radio Wireless Mesh Networks

Adword Auction

▶ Position Auction

Adwords Pricing

2007; Bu, Deng, Qi

TIAN-MING BU
Department of Computer Science & Engineering,
Fudan University, Shanghai, China

Problem Definition

The model studied here is the same as that which was first presented in [11] by Varian. For some keyword, $\mathcal{N} = \{1, 2, \ldots, N\}$, advertisers bid $\mathcal{K} = \{1, 2, \ldots, K\}$ advertisement slots ($K < N$) which will be displayed on the search result page from top to bottom. The higher the advertisement is positioned, the more conspicuous it is and the more clicks it receives. Thus for any two slots $k_1, k_2 \in \mathcal{K}$, if $k_1 < k_2$, then slot k_1's click-through rate (CTR) c_{k_1} is larger than c_{k_2}. That is, $c_1 > c_2 > \ldots > c_K$, from top to bottom, respectively. Moreover, each bidder $i \in \mathcal{N}$ has privately known information, v^i, which represents the expected return per click to bidder i.

According to each bidder i's submitted bid b^i, the auctioneer then decides how to distribute the advertisement slots among the bidders and how much they should pay per click. In particular, the auctioneer first sorts the bidders in decreasing order according to their submitted bids. Then the highest slot is allocated to the first bidder, the second highest slot is allocated to the second bidder, and so on. The last $N - K$ bidders would lose and get nothing. Finally, each winner would be charged on a per-click basis for the next bid in the descending bid queue. The losers would pay nothing.

Let b_k denote the kth highest bid in the descending bid queue and v_k the true value of the kth bidder in the descending queue. Thus if bidder i got slot k, i's payment would be $b_{k+1} \cdot c_k$. Otherwise, his payment would be zero. Hence, for any bidder $i \in \mathcal{N}$, if i were on slot $k \in \mathcal{K}$, his utility (payoff) could be represented as

$$u^i_k = (v^i - b_{k+1}) \cdot c_k \, .$$

Unlike one-round sealed-bid auctions where each bidder has only one chance to bid, the adword auction allows bidders to change their bids any time. Once bids are changed, the system refreshes the ranking automatically and instantaneously. Accordingly, all bidders' payment and utility are also recalculated. As a result, other bidders could then have an incentive to change their bids to increase their utility, and so on.

Definition 1 (Adword Pricing)
INPUT: the CTR for each slot, each bidder's expected return per click on his advertising.
OUTPUT: the stable states of this auction and whether any of these stable states can be reached from any initial states.

Key Results

Let $\mathbf{b}$ represent the bid vector $(b^1, b^2, \ldots, b^N)$. $\forall i \in \mathcal{N}$, $\mathcal{O}^i(\mathbf{b})$ denotes bidder i's place in the descending bid queue. Let $\mathbf{b}^{-i} = (b^1, \ldots, b^{i-1}, b^{i+1}, \ldots, b^N)$ denote the bids of all other bidders except i. $\mathcal{M}^i(\mathbf{b}^{-i})$ returns a set defined as

$$\mathcal{M}^i(\mathbf{b}^{-i}) = \arg \max_{b^i \in [0, v^i]} \left\{ u^i_{\mathcal{O}^i(b^i, \mathbf{b}^{-i})} \right\} . \tag{1}$$

Definition 2 (Forward-Looking Best-Response Function) Given $\mathbf{b}^{-i}$, suppose $\mathcal{O}^i(\mathcal{M}^i(\mathbf{b}^{-i}), \mathbf{b}^{-i}) = k$, then bidder i's forward-looking response function $\mathcal{F}^i(\mathbf{b}^{-i})$ is defined as

$$\mathcal{F}^i(\mathbf{b}^{-i}) = \begin{cases} v^i - \frac{c_k}{c_{k-1}}(v^i - b_{k+1}) & 2 \le k \le K\,, \\ v^i & k = 1 \text{ or } k > K\,. \end{cases} \tag{2}$$

Definition 3 (Forward-Looking Nash Equilibrium) A forward-looking best-response-function-based Nash equilibrium is a strategy profile $\hat{\mathbf{b}}$ such that

$$\forall i \in \mathcal{N}\,, \quad \hat{\mathbf{b}}^i \in \mathcal{F}^i(\hat{\mathbf{b}}^{-i})\,.$$

Definition 4 (Output Truthful [7,9]) For any instance of an adword auction and the corresponding equilibrium set $\mathcal{E}$, if $\forall \mathbf{e} \in \mathcal{E}$ and $\forall i \in \mathcal{N}$, $\mathcal{O}^i(\mathbf{e}) = \mathcal{O}^i(v^1, \dots, v^N)$, then the adword auction is *output truthful* on $\mathcal{E}$.

Theorem 5 *An adword auction is output truthful on* $\mathcal{E}_{\text{forward-looking}}$.

Corollary 6 *An adword auction has a unique forward-looking Nash equilibrium.*

Corollary 7 *Any bidder's payment under the forward-looking Nash equilibrium is equal to her payment under the VCG mechanism for the auction.*

Corollary 8 *For adword auctions, the auctioneer's revenue in a forward-looking Nash equilibrium is equal to her revenue under the VCG mechanism for the auction.*

Definition 9 (Simultaneous Readjustment Scheme) In a simultaneous readjustment scheme, all bidders participating in the auction will use forward-looking best-response function $\mathcal{F}$ to update their current bids simultaneously, which turns the current stage into a new stage. Then, based on the new stage, all bidders may update their bids again.

Theorem 10 *An adword auction may not always converge to a forward-looking Nash equilibrium under the simultaneous readjustment scheme even when the number of slots is 3. But the protocol converges when the number of slots is 2.*

Definition 11 (Round-Robin Readjustment Scheme) In the round-robin readjustment scheme, bidders update their biddings one after the other, according to the order of the bidder's number or the order of the slots.

Theorem 12 *An adword auction may not always converge to a forward-looking Nash equilibrium under the round-robin readjustment scheme even when the number of slots is 4. But the protocol converges when the number of slots is 2 or 3.*

```
if ($j = 0$) then
    exit
end if
Let $i$ be the ID of the bidder whose current bid is $b_j$ (and equivalently, $b^i$).
Let $h = \mathcal{O}^i(\mathcal{M}^i(\mathbf{b}^{-i}), \mathbf{b}^{-i})$.
Let $\mathcal{F}^i(\mathbf{b}^{-i})$ be the best response function value for Bidder $i$.
Re-sort the bid sequence. (So $h$ is the slot of the new bid $\mathcal{F}^i(\mathbf{b}^{-i})$ of Bidder $i$.)
if ($h < j$) then
    call Lowest-First($K, j, b_1, b_2, \cdots, b_N$),
else
    call Lowest-First($K, h-1, b_1, b_2, \cdots, b_N$)
end if
```

Adwords Pricing, Figure 1
Readjustment Scheme: Lowest-First($K, j, b_1, b_2, \cdots, b_N$)

Theorem 13 *Adword auctions converge to a forward-looking Nash equilibrium in finite steps with a* lowest-first *adjustment scheme.*

Theorem 14 *Adword auctions converge to a forward-looking Nash equilibrium with probability one under a randomized readjustment scheme.*

Applications

Online adword auctions are the fastest growing form of advertising on the Internet today. Many search engine companies such as Google and Yahoo! make huge profits on this kind of auction. Because advertisers can change their bids any time, such auctions can reduce advertisers' risk. Further, because the advertisement is only displayed to those people who are really interested in it, such auctions can reduce advertisers' investment and increase their return on investment.

For the same model, Varian [11] focuses on a subset of Nash equilibrium called *symmetric Nash equilibrium*, which can be formulated nicely and dealt with easily. Edelman et al. [8] study *locally envy-free* equilibrium, where no player can improve her payoff by exchanging bid with the player ranked one position above her. Coincidently, locally envy-free equilibrium is equal to symmetric Nash equilibrium proposed in [11]. Further, the revenue under the forward-looking Nash equilibrium is the same as the lower bound under Varian's symmetric Nash equilibrium and the lower bound under Edelman et al.'s locally envy-free equilibrium. In [6], Cary et al. also study the dynamic model's equilibrium and convergence based on the *balanced bidding strategy*, which is actually the same as the *forward-looking best-response function* in [4]. Cary et al. explore the convergence properties under two models, a *synchronous* model, which is the same as the *simultaneous readjustment scheme* in [4], and an *asynchronous* model, which is the same as the *randomized readjustment scheme* in [4].

In addition, there are other models for adword auctions. [1] and [5] study the model under which each bidder can submit a daily budget, even the maximum number of clicks per day, in addition to the price per click. Both [10] and [3] study bidders' behavior of bidding on several keywords. [2] studies a model whereby the advertiser not only submits a bid but additionally submits which positions he is going to bid for.

Open Problems

The speed of convergence remains open. Does the dynamic model converge in polynomial time under randomized readjustment scheme? Even more, are there other readjustment schemes that converge in polynomial time?

Cross References

► Multiple Unit Auctions with Budget Constraint
► Position Auction

Recommended Reading

1. Abrams, Z.: Revenue maximization when bidders have budgets. In: Proceedings of the 17th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA-06), Miami, FL 2006, pp. 1074–1082, ACM Press, New York (2006)
2. Aggarwal, G., Muthukrishnan, S., Feldman, J.: Bidding to the top: Vcg and equilibria of position-based auctions. http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0607117 (2006)
3. Borgs, C., Chayes, J., Etesami, O., Immorlica, N., Jain, K., Mahdian, M.: Bid optimization in online advertisement auctions. In: 2nd Workshop on Sponsored Search Auctions, in conjunction with the ACM Conference on Electronic Commerce (EC-06), Ann Arbor, MI, 2006
4. Bu, T.-M., Deng, X., Qi, Q.: Dynamics of strategic manipulation in ad-words auction. In: 3rd Workshop on Sponsored Search Auctions, in conjunction with WWW2007, Banff, Canada, 2007
5. Bu, T.-M., Qi, Q., Sun, A.W.: Unconditional competitive auctions with copy and budget constraints. In: Spirakis, P.G., Mavronicolas, M., Kontogiannis, S.C. (eds.) Internet and Network Economics, 2nd International Workshop, WINE 2006. Lecture Notes in Computer Science, vol. 4286, pp. 16–26, Patras, Greece, December 15–17. Springer, Berlin (2006)
6. Cary, M., Das, A., Edelman, B., Giotis, I., Heimerl, K., Karlin, A.R., Mathieu, C., Schwarz, M.: Greedy bidding strategies for keyword auctions. In: MacKie-Mason, J.K., Parkes, D.C., Resnick, P. (eds.) Proceedings of the 8th ACM Conference on Electronic Commerce (EC-2007), San Diego, California, USA, June 11–15 2007, pp. 262–271. ACM, New York (2007)
7. Chen, X., Deng, X., Liu, B.J.: On incentive compatible competitive selection protocol. In: Computing and Combinatorics, 12th Annual International Conference, COCOON 2006, Taipei, Taiwan, 15 August 2006. Lecture Notes in Computer Science, vol. 4112, pp. 13–22. Springer, Berlin (2006)
8. Edelman, B., Ostrovsky, M., Schwarz, M.: Internet advertising and the generalized second price auction: selling billions of dollars worth of dollars worth of keywords. In: 2nd Workshop on Sponsored Search Auctions, in conjunction with the ACM Conference on Electronic Commerce (EC-06), Ann Arbor, MI, June 2006
9. Kao, M.-Y., Li, X.-Y., Wang, W.: Output truthful versus input truthful: a new concept for algorithmic mechanism design (2006)
10. Kitts, B., Leblanc, B.: Optimal bidding on keyword auctions. Electronic Markets, Special issue: Innovative Auction Markets **14**(3), 186–201 (2004)
11. Varian, H.R.: Position auctions. Int. J. Ind. Organ. **25**(6), 1163–1178 (2007) http://www.sims.berkeley.edu/~hal/Papers/2006/position.pdf. Accessed 29 March 2006

Agreement

► Asynchronous Consensus Impossibility
► Consensus with Partial Synchrony
► Randomization in Distributed Computing

Algorithm DC-Tree for *k* Servers on Trees

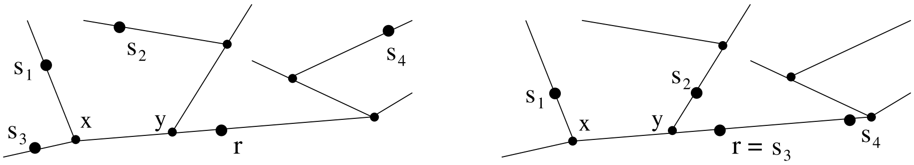
1991; Chrobak, Larmore

Marek Chrobak
Department of Computer Science,
University of California, Riverside, CA, USA

Problem Definition

In the *k-server problem*, one wishes to schedule the movement of k servers in a metric space $\mathbb{M}$, in response to a sequence $\varrho = r_1, r_2, \dots, r_n$ of *requests*, where $r_i \in \mathbb{M}$ for each i. Initially, all the servers are located at some point $r_0 \in \mathbb{M}$. After each request r_i is issued, one of the k servers must move to r_i. A *schedule* specifies which server moves to each request. The *cost* of a schedule is the total distance traveled by the servers, and our objective is to find a schedule with minimum cost.

In the *online* version of the k-server problem the decision as to which server to move to each request r_i must be made before the next request r_{i+1} is issued. In other words, the choice of this server is a function of requests



Algorithm DC-Tree for k Servers on Trees, Figure 1

Algorithm DC-TREE serving a request on r. The initial configuration is on the *left*; the configuration after the service is completed is on the *right*. At first, all servers are active. When server 3 reaches point x, server 1 becomes inactive. When server 3 reaches point y, server 2 becomes inactive

$r_1, r_2, \dots, r_i$. It is quite easy to see that in this online scenario it is not possible to guarantee an optimal schedule. The accuracy of online algorithms is often measured using competitive analysis. If $\mathcal{A}$ is an online k-server algorithm, denote by $cost_{\mathcal{A}}(\varrho)$ the cost of the schedule produced by $\mathcal{A}$ on a request sequence ϱ, and by $opt(\varrho)$ the cost of the optimal schedule. $\mathcal{A}$ is called *R-competitive* if $cost_{\mathcal{A}}(\varrho) \le R \cdot opt(\varrho) + B$, where B is a constant that may depend on $\mathbb{M}$ and r_0. The smallest such R is called the *competitive ratio* of $\mathcal{A}$. Of course, the smaller the R the better.

The k-server problem was introduced by Manasse, McGeoch, and Sleator [7,8], who proved that there is no online R-competitive algorithm for $R < k$, for any metric space with at least $k + 1$ points. They also gave a 2-competitive algorithm for $k = 2$ and formulated what is now known as the *k-server conjecture*, which postulates that there exists a k-competitive online algorithm for all k. Koutsoupias and Papadimitriou [5,6] proved that the so-called *work-function algorithm* has competitive ratio at most $2k - 1$, which to date remains the best upper bound known.

Efforts to prove the k-server conjecture led to discoveries of k-competitive algorithms for some restricted classes of metric spaces, including Algorithm DC-TREE for trees [4] presented in the next section. (See [1,2,3] for other examples.) A *tree* is a metric space defined by a connected acyclic graph whose edges are treated as line segments of arbitrary positive lengths. This metric space includes both the tree's vertices and the points on the edges, and the distances are measured along the (unique) shortest paths.

Key Results

Let $\mathbb{T}$ be a tree, as defined above. Given the current server configuration $S = \{s_1, \dots, s_k\}$, where s_j denotes the location of server j, and a request point r, the algorithm will move several servers, with one of them ending up on r. For two points $x, y \in \mathbb{T}$, let $[x, y]$ be the unique path from x to y in $\mathbb{T}$. A server j is called *active* if there is no other server in $[s_j, r] - \{s_j\}$, and j is the minimum-index server located on s_j (the last condition is needed only to break ties).

Algorithm DC-TREE

On a request r, move all active servers, continuously and with the same speed, towards r, until one of them reaches the request. Note that during this process some active servers may become inactive, in which case they halt. Clearly, the server that will arrive at r is the one that was closest to r at the time when r was issued. Figure 1 shows how DC-TREE serves a request r.

The competitive analysis of Algorithm DC-TREE is based on a potential argument. The cost of Algorithm DC-TREE is compared to that of an adversary who serves the requests with her own servers. Denoting by A the configuration of the adversary servers at a given step, define the potential by $\Phi = k \cdot D(S, A) + \sum_{i<j} d(s_i, s_j)$, where $D(S, A)$ is the cost of the minimum matching between S and A. At each step, the adversary first moves one of her servers to r. In this sub-step the potential increases by at most k times the increase of the adversary's cost. Then, Algorithm DC-TREE serves the request. One can show that then the sum of Φ and DC-TREE's cost does not increase. These two facts, by amortization over the whole request sequence, imply the following result [4]:

Theorem ([4]) *Algorithm* DC-TREE *is k-competitive on trees.*

Applications

The k-server problem is an abstraction of various scheduling problems, including emergency crew scheduling, caching in multilevel memory systems, or scheduling head movement in 2-headed disks. Nevertheless, due to its abstract nature, the k-server problem is mainly of theoretical interest.

Algorithm DC-TREE can be applied to other spaces by "embedding" them into trees. For example, a uniform metric space (with all distances equal 1) can be represented by a star with arms of length 1/2, and thus Algorithm DC-TREE can be applied to those spaces. This also immediately gives a k-competitive algorithm for the *caching problem*, where the objective is to manage a two-level memory system consisting of a large main memory and a cache that can store up to k memory items. If an item is in the cache, it can be accessed at cost 0, otherwise it costs 1 to read it from the main memory. This caching problem can be thought of as the k-server problem in a uniform metric space where the server positions represent the items residing in the cache. This idea can be extended further to the *weighted caching* [3], which is a generalization of the caching problem where different items may have different costs. In fact, if one can embed a metric space $\mathbb{M}$ into a tree with distortion bounded by δ, then Algorithm DC-TREE yields a δk-competitive algorithm for $\mathbb{M}$.

Open Problems

The k-server conjecture – whether there is a k-competitive algorithm for k servers in any metric space – remains open. It would be of interest to prove it for some natural special cases, for example the plane, either with the Euclidean or Manhattan metric. (A k-competitive algorithm for the Manhattan plane for $k = 2, 3$ servers is known [1], but not for $k \geq 4$.)

Very little is known about online *randomized* algorithms for k-servers. In fact, even for $k = 2$ it is not known if there is a randomized algorithm with competitive ratio smaller than 2.

Cross References

- ▶ Deterministic Searching on the Line
- ▶ Generalized Two-Server Problem
- ▶ Metrical Task Systems
- ▶ Online Paging and Caching
- ▶ Paging
- ▶ Work-Function Algorithm for k Servers

Recommended Reading

1. Bein, W., Chrobak, M., Larmore, L.L.: The 3-server problem in the plane. Theor. Comput. Sci. **287**, 387–391 (2002)
2. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
3. Chrobak, M., Karloff, H., Payne, T.H., Vishwanathan, S.: New results on server problems. SIAM J. Discret. Math. **4**, 172–181 (1991)
4. Chrobak, M., Larmore, L.L.: An optimal online algorithm for k servers on trees. SIAM J. Comput. **20**, 144–148 (1991)
5. Koutsoupias, E., Papadimitriou, C.: On the k-server conjecture. In: Proc. 26th Symp. Theory of Computing (STOC), pp. 507–511. ACM (1994)
6. Koutsoupias, E., Papadimitriou, C.: On the k-server conjecture. J. ACM **42**, 971–983 (1995)
7. Manasse, M., McGeoch, L.A., Sleator, D.: Competitive algorithms for online problems. In: Proc. 20th Symp. Theory of Computing (STOC), pp. 322–333. ACM (1988)
8. Manasse, M., McGeoch, L.A., Sleator, D.: Competitive algorithms for server problems. J. Algorithms **11**, 208–230 (1990)

Algorithmic Cooling

1999; Schulman, Vazirani
2002; Boykin, Mor, Roychowdhury, Vatan, Vrijen

TAL MOR
Department of Computer Science, Technion, Haifa, Israel

Keywords and Synonyms

Algorithmic cooling of spins; Heat-bath algorithmic cooling

Problem Definition

The fusion of concepts taken from the fields of quantum computation, data compression, and thermodynamics, has recently yielded novel algorithms that resolve problems in nuclear magnetic resonance and potentially in other areas as well; algorithms that "cool down" physical systems.

- A leading candidate technology for the construction of quantum computers is Nuclear Magnetic Resonance (NMR). This technology has the advantage of being well-established for other purposes, such as chemistry and medicine. Hence, it does not require new and exotic equipment, in contrast to ion traps and optical lattices, to name a few. However, when using standard NMR techniques (not only for quantum computing purposes) one has to live with the fact that the state can only be initialized in a very noisy manner: The particles' spins point in mostly random directions, with only a tiny bias towards the desired state.

  The key idea of Schulman and Vazirani [13] is to combine the tools of both data compression and quantum computation, to suggest a *scalable* state initialization process, a "molecular-scale heat engine". Based on Schulman and Vazirani's method, Boykin, Mor, Roychowdhury, Vatan, and Vrijen [2] then developed a new process, "heat-bath algorithmic cooling", to significantly improve the state initialization process, by opening the system to the environment. Strikingly, this offered a way to put to good use the phenomenon of decoherence, which is usually considered to be the villain in quantum computation. These two methods are now sometimes called "closed-system" (or "reversible") algorithmic cooling, and "open-system" algorithmic cooling, respectively.

- The far-reaching consequence of this research lies in the possibility of reaching beyond the potential implementation of remote-future quantum computing devices. An efficient technique to generate ensembles of spins that are highly polarized by external magnetic fields is considered to be a Holy Grail in NMR spectroscopy. Spin-half nuclei have steady-state polarization biases that increase inversely with temperature; therefore, spins exhibiting polarization biases above their thermal-equilibrium biases are considered *cool*. Such cooled spins present an improved signal-to-noise ratio if used in NMR spectroscopy or imaging.

  Existing spin-cooling techniques are limited in their efficiency and usefulness. Algorithmic cooling is a promising new spin-cooling approach that employs data compression methods in *open systems*. It reduces the entropy of spins to a point far beyond Shannon's entropy bound on reversible entropy manipulations, thus increasing their polarization biases. As a result, it is conceivable that the open-system algorithmic cooling technique could be harnessed to improve on *current uses of NMR* in areas such as chemistry, material science, and even medicine, since NMR is at the basis of MRI – Magnetic Resonance Imaging.

Basic Concepts

Loss-Less in-Place Data Compression Given a bit-string of length n, such that the probability distribution is known and far enough from the uniform distribution, one can use data compression to generate a shorter string, say of m bits, such that the entropy of each bit is much closer to one. As a simple example, consider a four-bit-string which is distributed as follows; $p_{0001} = p_{0010} = p_{0100} = p_{1000} = 1/4$, with p_i the probability of the string i. The probability of any other string value is exactly zero, so the probabilities sum up to one. Then, the bit-string can be compressed, via a loss-less compression algorithm, into a 2-bit string that holds the binary description of the location of "1" in the above four strings. As the probabilities of all these strings are zero, one can also envision a similar process that generates an output which is of the same length n as the input, but such that the entropy is compressed via a loss-less, in-place, data compression into the last two bits. For instance, logical gates that operate on the bits can perform the permutation 0001 → 0000, 0010 → 0001, 0100 → 0010 and 1000 → 0011, while the other input strings transform to output strings in which the two most significant bits are not zero; for instance 1100 → 1010. One can easily see that the entropy is now fully concentrated on the two least significant bits, which are useful in data compression, while the two most significant bits have zero entropy.

In order to gain some intuition about the design of logical gates that perform entropy manipulations, one can look at a closely related scenario which was first considered by von Neumann. He showed a method to extract fair coin flips, given a biased coin; he suggested taking a pair of biased coin flips, with results a and b, and using the value of a **conditioned on** $a \neq b$. A simple calculation shows that $a = 0$ and $a = 1$ are now obtained with equal probabilities, and therefore the entropy of coin a is increased in this case to 1. The opposite case, the probability distribution of a given that $a = b$, results in a highly determined coin flip; namely, a (conditioned) coin-flip with a higher bias or lower entropy. A gate that flips the value of b if (and only if) $a = 1$ is called a Controlled-NOT gate. If after applying such a gate $b = 1$ is obtained, this means that $a \neq b$ prior to the gate operation, thus now the entropy of a is 1. If, on the other hand, after applying such a gate $b = 0$ is obtained, this means that $a = b$ prior to the gate operation, thus the entropy of a is now lower than its initial value.

Spin Temperature, Polarization Bias, and Effective Cooling In physics, two-level systems, namely systems that possess only binary values, are useful in many ways. Often it is important to initialize such systems to a pure state '0' or to a probability distribution which is as close as possible to a pure state '0'. In these physical two-level systems a data compression process that brings some of them closer to a pure state can be considered as "cooling". For quantum two-level systems there is a simple connection between temperature, entropy, and population probability. The population-probability difference between these two levels is known as the polarization bias, ϵ. Consider a single spin-half particle – for instance a hydrogen nucleus – in a constant magnetic field. At equilibrium with a thermal heat-bath the probability of this spin to be up or down (i. e., parallel or anti-parallel to the field direction) is given by: $p_{\uparrow} = \frac{1+\epsilon}{2}$, and $p_{\downarrow} = \frac{1-\epsilon}{2}$. The entropy H of the spin is $H(\text{single-bit}) = H(1/2 + \epsilon/2)$ with $H(P) \equiv -P \log_2 P - (1-P)\log_2(1-P)$ measured in bits. The two pure states of a spin-half nucleus are commonly written as $|\uparrow\rangle \equiv$'0' and $|\downarrow\rangle \equiv$'1'; the $|\rangle$ notation will be clarified elsewhere[1]. The polarization bias of the spin at thermal equilibrium is given by $\epsilon = p_{\uparrow} - p_{\downarrow}$. For such a physical system the bias is obtained via a quantum statistical mechanics argument, $\epsilon = \tanh\left(\frac{\hbar\gamma B}{2K_B T}\right)$, where $\hbar$ is Planck's constant, B is the magnetic field, γ is the

[1] Quantum Computing entries in this encyclopedia, e.g. ► Quantum Dense Coding

particle-dependent gyromagnetic constant[2], K_B is Boltzman's coefficient, and T is the thermal heat-bath temperature. For high temperatures or small biases $\epsilon \approx \frac{\hbar\gamma B}{2K_B T}$, thus the bias is inversely proportional to the temperature. Typical values of ϵ for spin-half nuclei at room temperature (and magnetic field of ~ 10 Tesla) are 10^{-5}–10^{-6}, and therefore most of the analysis here is done under the assumption that $\epsilon \ll 1$. The spin temperature at equilibrium is thus $T = \frac{\text{Const}}{\epsilon}$, and its (Shannon) entropy is $H = 1 - (\epsilon^2/\ln 4)$.

A spin temperature out of thermal equilibrium is still defined via the same formulas. Therefore, when a system is moved away from thermal equilibrium, achieving a greater polarization bias is equivalent to cooling the spins *without cooling the system*, and to decreasing their entropy. The process of increasing the bias (reducing the entropy) without decreasing the temperature of the thermal-bath is known as "effective cooling". After a typical period of time, termed the thermalization time or relaxation time, the bias will gradually revert to its thermal equilibrium value; yet during this process, typically in the order of seconds, the effectively-cooled spin may be used for various purposes as described in Sect. "Applications".

Consider a molecule that contains n adjacent spin-half nuclei arranged in a line; these form the bits of the string. These spins are initially at thermal equilibrium due to their interaction with the environment. At room temperature the bits at thermal equilibrium are not correlated to their neighbors on the same string: More precisely, the correlation is very small and can be ignored. Furthermore, in a liquid state one can also neglect the interaction between strings (between molecules). It is convenient to write the probability distribution of a single spin at thermal equilibrium using the "density matrix" notation

$$\rho_\epsilon = \begin{pmatrix} p_\uparrow & 0 \\ 0 & p_\downarrow \end{pmatrix} = \begin{pmatrix} (1+\epsilon)/2 & 0 \\ 0 & (1-\epsilon)/2 \end{pmatrix} , \qquad (1)$$

since these two-level systems are of a quantum nature (namely, these are quantum bits – qubits), and in general, can also have states other than just a classical probability distribution over '0' and '1'. The classical case will now be considered, where ρ contains only diagonal elements and these describe a conventional probability distribution. At thermal equilibrium, the state of $n = 2$ uncorrelated qubits that have the same polarization bias is described by the density matrix $\rho_{\text{init}}^{\{n=2\}} = \rho_\epsilon \otimes \rho_\epsilon$, where $\otimes$ means tensor product. The probability of the state '00', for instance, is then $(1+\epsilon)/2 \times (1+\epsilon)/2 = (1+\epsilon)^2/4$ (etc.). Similarly, the initial state of an n-qubit system of this type, at thermal equilibrium, is

$$\rho_{\text{init}}^{\{n\}} = \rho_\epsilon \otimes \rho_\epsilon \otimes \cdots \otimes \rho_\epsilon . \qquad (2)$$

This state represents a thermal probability distribution, such that the probability of the classical state '000...0' is $P_{000\ldots0} = (1+\epsilon_0)^n/2^n$, etc. In reality, the initial bias is not the same on each qubit[3], but as long as the differences between these biases are small (e. g., all qubits are of the same nucleus), these differences can be ignored in a discussion of an idealized scenario.

Key Results

Molecular Scale Heat Engines

Schulman and Vazirani (SV) [13] identified the importance of in-place loss-less data compression and of the low-entropy bits created in that process: Physical two-level systems (e. g., spin-half nuclei) may be similarly cooled by data compression algorithms. SV analyzed the cooling of such a system using various tools of data compression. A loss-less compression of an n-bit binary string distributed according to the thermal equilibrium distribution, Eq. (2), is readily analyzed using information-theoretical tools: In an ideal compression scheme (not necessarily realizable), with sufficiently large n, all randomness – and hence all the entropy – of the bit string is transferred to $n - m$ bits; the remaining m bits are thus left, with extremely high probability, at a known deterministic state, say the string '000...0'. The entropy H of the entire system is $H(\text{system}) = nH(\text{single} - \text{bit}) = nH(1/2 + \epsilon/2)$. Any compression scheme cannot decrease this entropy, hence Shannon's source coding entropy bound yields $m \le n[1 - H(1/2 + \epsilon/2)]$. A simple leading-order calculation shows that m is bounded by (approximately) $\frac{\epsilon^2}{2\ln 2} n$ for small values of the initial bias ϵ. Therefore, with typical $\epsilon \sim 10^{-5}$, molecules containing an order of magnitude of 10^{10} spins are required to cool a single spin close to zero temperature.

Conventional methods for NMR quantum computing are based on unscalable state-initialization schemes [5,9] (e. g., the "pseudo-pure-state" approach) in which the signal-to-noise ratio falls exponentially with n, the number of spins. Consequently, these methods are deemed inappropriate for future NMR quantum computers. SV [13] were first to employ tools of information theory to address

[2] This constant, γ, is thus responsible for the difference in equilibrium polarization bias [e. g., a hydrogen nucleus is 4 times more polarized than a carbon isotope ^{13}C nucleus, but about 10^3 less polarized than an electron spin].

[3] Furthermore, individual addressing of each spin during the algorithm requires a slightly different bias for each.

the scaling problem; they presented a compression scheme in which the number of cooled spins scales well (namely, a constant times n). SV also demonstrated a scheme approaching Shannon's entropy bound, for very large n. They provided detailed analyses of three cooling algorithms, each useful for a different regime of ϵ values.

Some ideas of SV were already explored a few years earlier by Sørensen [14], a physical chemist who analyzed effective cooling of spins. He considered the entropy of several spin systems and the limits imposed on cooling these systems by polarization transfer and more general polarization manipulations. Furthermore, he considered spin-cooling processes in which only unitary operations were used, wherein unitary matrices are applied to the density matrices; such operations are realizable, at least from a conceptual point of view. Sørensen derived a stricter bound on unitary cooling, which today bears his name. Yet, unlike SV, he did not infer the connection to data compression or advocate compression algorithms.

SV named their concept "molecular-scale heat engine". When combined with conventional polarization transfer (which is partially similar to a SWAP gate between two qubits), the term "reversible polarization compression (RPC)" to be more descriptive.

Heat-Bath Algorithmic Cooling

The next significant development came when Boykin, Mor, Roychowdhury, Vatan and Vrijen, (hereinafter referred to as BMRVV), invented a new spin-cooling technique, which they named *Algorithmic cooling* [2], or more specifically, heat-bath algorithmic cooling in which the use of controlled interactions with a heat bath enhances the cooling techniques much further. Algorithmic Cooling (AC) expands the effective cooling techniques by exploiting entropy manipulations in *open systems*. It combines RPC steps[4] with fast relaxation (namely, thermalization) of the *hotter spins*, as a way of pumping entropy outside the system and cooling the system *much beyond Shannon's entropy bound*. In order to pump entropy out of the system, AC employs regular spins (here called computation spins) together with rapidly relaxing spins. The latter are auxiliary spins that return to their thermal equilibrium state very rapidly. These spins have been termed "reset spins", or, equivalently, reset bits. The controlled interactions with the heat bath are generated by polarization transfer or by standard algorithmic techniques (of data compression) that transfer the entropy onto the reset spins which then lose this excess entropy into the environment.

The ratio $R_{\text{relax-times}}$, between the relaxation time of the computation spins and the relaxation time of the reset spins, must satisfy $R_{\text{relax-times}} \gg 1$. This condition is vital if one wishes to perform many cooling steps on the system to obtain significant cooling.

From a pure information-theoretical point of view, it is legitimate to assume that the only restriction on ideal RPC steps is Shannon's entropy bound; then the equivalent of Shannon's entropy bound, when an ideal open-system AC is used, is that all computation spins can be cooled down to zero temperature, that is to $\epsilon = 1$. Proof. – repeat the following till the entropy of all computation spins is exactly zero: (i) push entropy from computation spins into reset spins; (ii) let the reset spins cool back to room temperature. Clearly, each application of step (i), except the last one, pushes the same amount of entropy onto the reset spins, and then this entropy is removed from the system in step (ii). Of course, a realistic scenario must take other parameters into account such as finite relaxation-time ratios, realistic environment, and physical operations on the spins. Once this is done, cooling to zero temperature is no longer attainable. While finite relaxation times and a realistic environment are system dependent, the constraint of using physical operations is conceptual.

BMRVV therefore pursued an algorithm that follows some physical rules, it is performed by unitary operations and reset steps, and still bypass Shannon's entropy bound, by far. The BMRVV cooling algorithm obtains significant cooling beyond that entropy bound by making use of very long molecules bearing hundreds or even thousands of spins, because its analysis relies on the law of large numbers.

Practicable Algorithmic Cooling

The concept of algorithmic cooling then led to practicable algorithms [8] for cooling *small molecules*. In order to see the impact of practicable algorithmic cooling, it is best to use a different variant of the entropy bound. Consider a system containing n spin-half particles with total entropy higher than $n-1$, so that there is no way to cool even one spin to zero temperature. In this case, the entropy bound is a result of the compression of the entropy into $n-1$ fully-random spins, so that the remaining entropy on the last spin is minimal. The entropy of the remaining single spin satisfies $H(\text{single}) \geq 1 - n\epsilon^2/\ln 4$, thus, at most, its polarization can be improved to

$$\epsilon_{\text{final}} \leq \epsilon\sqrt{n}\ . \tag{3}$$

[4] When the entire process is RPC, namely, any of the processes that follow SV ideas, one can refer to it as reversible AC or closed-system AC, rather than as RPC.

The practicable algorithmic cooling (PAC), suggested by Fernandez, Lloyd, Mor, and Roychowdhury in [8], indicated potential for a near-future application to NMR spectroscopy. In particular, it presented an algorithm named PAC2 which uses any (odd) number of spins n, such that one of them is a reset spin, and $(n-1)$ are computation spins. PAC2 cools the spins such that the coldest one can (approximately) reach a bias amplification by a factor of $(3/2)^{(n-1)/2}$. The approximation is valid as long as the final bias $(3/2)^{(n-1)/2}\epsilon$ is much smaller than 1. Otherwise, a more precise treatment must be done. This proves an exponential advantage of AC over the best possible reversible AC, as these reversible cooling techniques, e. g., of [13,14], are limited to improve the bias by no more than a factor of $\sqrt{n}$. PAC can be applied for small n (e. g., in the range of 10–20), and therefore it is potentially suitable for near-future applications [6,8,10] in chemical and biomedical usages of NMR spectroscopy.

It is important to note that in typical scenarios the initial polarization bias of a reset spin is higher than that of a computation spin. In this case, the bias amplification factor of $(3/2)^{(n-1)/2}$ is relative to the larger bias, that of the reset spin.

Exhaustive Algorithmic Cooling

Next, AC was analyzed, wherein the cooling steps (reset and RPC) are repeated an arbitrary number of times. This is actually an idealization where an unbounded number of reset and logic steps can be applied without error or decoherence, while the computation qubits do not lose their polarization biases. Fernandez [7] considered two computation spins and a single reset spin (the least significant bit, namely the qubit at the right in the tensor-product density-matrix notation) and analyzed optimal cooling of this system. By repeating the reset and compression exhaustively, he realized that the bound on the final biases of the three spins is approximately $\{2, 1, 1\}$ in units of ϵ, the polarization bias of the reset spin.

Mor and Weinstein generalized this analysis further and found that $n-1$ computation spins and a single reset spin can be cooled (approximately) to biases according to the Fibonacci series: {... 34, 21, 13, 8, 5, 3, 2, 1, 1}. The computation spin that is furthest from the reset spin can be cooled up to the relevant Fibonacci number F_n. That approximation is valid as long as the largest term times ϵ is still much smaller than 1. Schulman then suggested the "partner pairing algorithm" (PPA) and proved the optimality of the PPA among all *classical and quantum* algorithms. These two algorithms, the Fibonacci AC and the PPA, led to two joint papers [11,12], where upper and lower bounds on AC were also obtained. The PPA is defined as follows; repeat these two steps until cooling sufficiently close to the limit: (a) RESET – applied to a reset spin in a system containing $n-1$ computation spins and a single (the LSB) reset spin. (b) SORT – a permutation that sorts the 2^n diagonal elements of the density matrix by decreasing order, so that the MSB spin becomes the coldest. Two important theorems proven in [12] are: 1. Lower bound: When $\epsilon 2^n \gg 1$ (namely, for long enough molecules), Theorem 3 in [12] promises that $n - \log(1/\epsilon)$ cold qubits can be extracted. This case is relevant for scalable NMR quantum computing. 2. Upper bound: Section 4.2 in [12] proves the following theorem: No algorithmic cooling method can increase the probability of any basis state to above $\min\{2^{-n} e^{2^n\epsilon}, 1\}$, wherein the initial configuration is the completely mixed state (the same is true if the initial state is a thermal state).

More recently, Elias, Fernandez, Mor, and Weinstein [6] analyzed more closely the case of $n < 15$ (at room temperature), where the coldest spin (at all stages) still has a polarization bias much smaller than 1. This case is most relevant for near-future applications in NMR spectroscopy. They generalized the Fibonacci-AC to algorithms yielding higher-term Fibonacci series, such as the tri-bonacci (also known as 3-term Fibonacci series), {... 81, 44, 24, 13, 7, 4, 2, 1, 1}, etc. The ultimate limit of these multi-term Fibonacci series is obtained when each term in the series is the sum of all previous terms. The resulting series is precisely the exponential series {... 128, 64, 32, 16, 8, 4, 2, 1, 1}, so the coldest spin is cooled by a factor of 2^{n-2}. Furthermore, a leading order analysis of the upper bound mentioned above (Section 4.2 in [12]) shows that no spin can be cooled beyond a factor of 2^{n-1}; see Corollary 1 in [6].

Applications

The two major far-future and near-future applications are already described in Sect. "Problem Definition". It is important to add here that although the specific algorithms analyzed so far for AC are usually classical, their practical implementation via an NMR spectrometer must be done through analysis of universal quantum computation, using the specific gates allowed in such systems. Therefore, AC could yield the first near-future application of quantum computing devices.

AC may also be useful for cooling various other physical systems, since state initialization is a common problem in physics in general and in quantum computation in particular.

Open Problems

A main open problem in practical AC is technological; can the ratio of relaxation times be increased so that many cooling steps may be applied onto relevant NMR systems? Other methods, for instance a spin-diffusion mechanism [1], may also be useful for various applications.

Another interesting open problem is whether the ideas developed during the design of AC can also lead to applications in classical information theory.

Experimental Results

Various ideas of AC had already led to several experiments using 3–4 qubit quantum computing devices: 1. An experiment [4] that implemented a single RPC step. 2. An experiment [3] in which entropy-conservation bounds (which apply in any closed system) were bypassed. 3. A full AC experiment [1] that includes the initialization of three carbon nuclei to the bias of a hydrogen spin, followed by a single compression step on these three carbons.

Cross References

► Dictionary-Based Data Compression
► Quantum Algorithm for Factoring
► Quantum Algorithm for the Parity Problem
► Quantum Dense Coding
► Quantum Key Distribution

Recommended Reading

1. Baugh, J., Moussa, O., Ryan, C.A., Nayak, A., Laflamme, R.: Experimental implementation of heat-bath algorithmic cooling using solid-state nuclear magnetic resonance. Nature **438**, 470–473 (2005)
2. Boykin, P.O., Mor, T., Roychowdhury, V., Vatan, F., Vrijen, R.: Algorithmic cooling and scalable NMR quantum computers. Proc. Natl. Acad. Sci. **99**, 3388–3393 (2002)
3. Brassard, G., Elias, Y., Fernandez, J.M., Gilboa, H., Jones, J.A., Mor, T., Weinstein, Y., Xiao, L.: Experimental heat-bath cooling of spins. Submitted to Proc. Natl. Acad. Sci. USA. See also quant-ph/0511156 (2005)
4. Chang, D.E., Vandersypen, L.M.K., Steffen, M.: NMR implementation of a building block for scalable quantum computation. Chem. Phys. Lett. **338**, 337–344 (2001)
5. Cory, D.G., Fahmy, A.F., Havel, T.F.: Ensemble quantum computing by NMR spectroscopy. Proc. Natl. Acad. Sci. **94**, 1634–1639 (1997)
6. Elias, Y., Fernandez, J.M., Mor, T., Weinstein, Y.: Optimal algorithmic cooling of spins. Isr. J. Chem. **46**, 371–391 (2006), also in: Ekl, S. et al. (eds.) Lecture Notes in Computer Science, Volume 4618, pp. 2–26. Springer, Berlin (2007), Unconventional Computation. Proceedings of the Sixth International Conference UC2007 Kingston, August 2007
7. Fernandez, J.M.: De computatione quantica. Dissertation, University of Montreal (2004)
8. Fernandez, J.M., Lloyd, S., Mor, T., Roychowdhury V.: Practicable algorithmic cooling of spins. Int. J. Quant. Inf. **2**, 461–477 (2004)
9. Gershenfeld, N.A., Chuang, I.L.: Bulk spin-resonance quantum computation. Science **275**, 350–356 (1997)
10. Mor, T., Roychowdhury, V., Lloyd, S., Fernandez, J.M., Weinstein, Y.: Algorithmic cooling. US Patent 6,873,154 (2005)
11. Schulman, L.J., Mor, T., Weinstein, Y.: Physical limits of heat-bath algorithmic cooling. Phys. Rev. Lett. **94**, 120501, pp. 1–4 (2005)
12. Schulman, L.J., Mor, T., Weinstein, Y.: Physical limits of heat-bath algorithmic cooling. SIAM J. Comput. **36**, 1729–1747 (2007)
13. Schulman, L.J., Vazirani, U.: Molecular scale heat engines and scalable quantum computation. Proc. 31st ACM STOC, Symp. Theory of Computing,pp. 322–329 Atlanta, 01–04 May 1999
14. Sørensen, O.W.: Polarization transfer experiments in high-resolution NMR spectroscopy. Prog. Nuc. Mag. Res. Spect. **21**, 503–569 (1989)

Algorithmic Mechanism Design

1999; Nisan, Ronen

RON LAVI
Faculty of Industrial Engineering and Management,
Technion, Haifa, Israel

Problem Definition

Mechanism design is a sub-field of economics and game theory that studies the construction of social mechanisms in the presence of selfish agents. The nature of the agents dictates a basic contrast between the social planner, that aims to reach a socially desirable outcome, and the agents, that care only about their own private utility. The underlying question is how to incentivize the agents to cooperate, in order to reach the desirable social outcomes.

In the Internet era, where computers act and interact on behalf of selfish entities, the connection of the above to algorithmic design suggests itself: suppose that the input to an algorithm is kept by selfish agents, who aim to maximize their own utility. How can one design the algorithm so that the agents will find it in their best interest to cooperate, and a close-to-optimal outcome will be outputted? This is different than classic distributed computing models, where agents are either "good" (meaning obedient) or "bad" (meaning faulty, or malicious, depending on the context). Here, no such partition is possible. It is simply assumed that all agents are utility maximizers. To illustrate this, let us describe a motivating example:

A Motivating Example: Shortest Paths

Given a weighted graph, the goal is to find a shortest path (with respect to the edge weights) between a given source and target nodes. Each edge is controlled by a selfish entity, and the weight of the edge, w_{e} is private information of that edge. If an edge is chosen by the algorithm to be included in the shortest path, it will incur a cost which is minus its weight (the cost of communication). Payments to the edges are allowed, and the total utility of an edge that participates in the shortest path and gets a payment p_{e} is assumed to be $u_{\mathrm{e}} = p_{\mathrm{e}} - w_{\mathrm{e}}$. Notice that the shortest path is *with respect to the true weights of the agents, although these are not known to the designer.*

Assuming that each edge will act in order to maximize its utility, how can one choose the path and the payments? One option is to ignore the strategic issue all together, ask the edges to simply report their weights, and compute the shortest path. In this case, however, an edge dislikes being selected, and will therefore prefer to report a very high weight (much higher than its true weight) in order to decrease the chances of being selected. Another option is to pay each selected edge its reported weight, or its reported weight plus a small fixed "bonus". However in such a case all edges will report lower weights, as being selected will imply a positive gain.

Although this example is written in an algorithmic language, it is actually a mechanism design problem, and the solution, which is now a classic, was suggested in the 70's. The chapter continues as follows: First, the abstract formulation for such problems is given, the classic solution from economics is described, and its advantages and disadvantages for algorithmic purposes are discussed. The next section then describes the new results that algorithmic mechanism design offers.

Abstract Formulation

The framework consists of a set A of alternatives, or outcomes, and n players, or agents. Each player i has a valuation function $v_i : A \to \Re$ that assigns a value to each possible alternative. This valuation function belongs to a domain V_i of all possible valuation functions. Let $V = V_1 \times \cdots \times V_n$, and $V_{-i} = \prod_{j \neq i} V_j$. Observe that this generalizes the shortest path example of above: A is all the possible $s - t$ paths in the given graph, $v_{\mathrm{e}}(a)$ for some path $a \in A$ is either $-w_{\mathrm{e}}$ (if $e \in a$) or zero.

A *social choice function* $f : V \to A$ assigns a socially desirable alternative to any given profile of players' valuations. This parallels the notion of an algorithm. A *mechanism* is a tuple $M = (f, p_1, \ldots, p_n)$, where f is a social choice function, and $p_i : V \to \Re$ (for $i = 1, \ldots, n$) is the price charged from player i. The interpretation is that the social planner asks the players to reveal their true valuations, chooses the alternative according to f as if the players have indeed acted truthfully, and in addition rewards/punishes the players with the prices. These prices should induce "truthfulness" in the following strong sense: no matter what the other players declare, it is always in the best interest of player i to reveal her true valuation, as this will maximize her utility. Formally, this translates to:

Definition 1 (Truthfulness) M is "truthful" (in dominant strategies) if, for any player i, any profile of valuations of the other players $v_{-i} \in V_{-i}$, and any two valuations of player $i v_i, v'_i \in V_i$,

$$v_i(a) - p_i(v_i, v_{-i}) \geq v_i(b) - p_i(v'_i, v_{-i})$$

where $f(v_i, v_{-i}) = a$ and $f(v'_i, v_{-i}) = b$.

Truthfulness is quite strong: a player need not know anything about the other players, even not that they are rational, and still determine the best strategy for her. Quite remarkably, there exists a truthful mechanism, even under the current level of abstraction. This mechanism suits all problem domains, where the social goal is to maximize the "social welfare":

Definition 2 (Social welfare maximization) A social choice function $f : V \to A$ maximizes the social welfare if $f(v) \in \operatorname{argmax}_{a \in A} \sum_i v_i(a)$, for any $v \in V$.

Notice that the social goal in the shortest path domain is indeed welfare maximization, and, in general, this is a natural and important economic goal. Quite remarkably, there exists a general technique to construct truthful mechanisms that implement this goal:

Theorem 1 (Vickrey–Clarke–Groves (VCG)) *Fix any alternatives set A and any domain V, and suppose that $f : V \to A$ maximizes the social welfare. Then there exist prices p such that the mechanism (f, p) is truthful.*

This gives "for free" a solution to the shortest path problem, and to many other algorithmic problems. The great advantage of the VCG scheme is its generality: it suits *all* problem domains. The disadvantage, however, is that the method is tailored to social welfare maximization. This turns out to be restrictive, especially for algorithmic and computational settings, due to several reasons: (i) different algorithmic goals: the algorithmic literature considers a variety of goals, including many that cannot be translated to welfare maximization. VCG does not help us in such cases. (ii) computational complexity: even if the goal is welfare maximization, in many settings achieving exactly the optimum is computationally hard. The CS discipline usually overcomes this by using approximation algorithms, but VCG will not work with such algorithm – reaching exact optimality is a necessary requirement of VCG. (iii) different algorithmic models: common CS models change "the basic setup", hence cause unexpected difficulties when one tries to use VCG (for example, an online model, where the input is revealed over time; this is common in CS, but changes the implicit setting that VCG requires). This is true even if welfare maximization is still the goal.

Answering any one of these difficulties requires the design of a non-VCG mechanism. What analysis tools should be used for this purpose? In economics and classic mechanism design, average-case analysis, that relies on the knowledge of the underlying distribution, is the standard. Computer science, on the other hand, usually prefers to avoid strong distributional assumptions, and to use worst-case analysis. This difference is another cause to the uniqueness of the answers provided by algorithmic mechanism design. Some of the new results that have emerged as a consequence of this integration between Computer Science and Economics is next described. Many other research topics that use the tools of algorithmic mechanism design are described in the entries on Adword Pricing, Competitive Auctions, False Name Proof Auctions, Generalized Vickrey Auction, Incentive Compatible Ranking, Mechanism for One Parameter Agents Single Buyer/Seller, Multiple Item Auctions, Position Auctions, and Truthful Multicast.

There are two different but closely related research topics that should be mentioned in the context of this entry. The first is the line of works that studies the "price of anarchy" of a given system. These works analyze *existing* systems, trying to quantify the loss of social efficiency due to the selfish nature of the participants, while the approach of algorithmic mechanism design is to understand how new systems should be designed. For more details on this topic the reader is referred to the entry on Price of Anarchy. The second topic regards the algorithmic study of various equilibria computation. These works bring computational aspects into economics and game theory, as they ask what equilibria notions are reasonable to assume, if one requires computational efficiency, while the works described here bring game theory and economics into computer science and algorithmic theory, as they ask what algorithms are reasonable to design, if one requires the resilience to selfish behavior. For more details on this topic the reader is referred (for example) to the entry on Algorithms for Nash Equilibrium and to the entry on General Equilibrium.

Key Results

Problem Domain 1: Job Scheduling

Job scheduling is a classic algorithmic setting: n jobs are to be assigned to m machines, where job j requires processing time p_{ij} on machine i. In the game-theoretic setting, it is assumed that each machine i is a selfish entity, that incurs a cost p_{ij} from processing job j. Note that the payments in this setting (and in general) may be negative, offsetting such costs. A popular algorithmic goal is to assign jobs to machines in order to minimize the "makespan": $\max_i \sum_{j \text{ is assigned to } i} p_{ij}$. This is different than welfare maximization, which translates in this setting to the minimization of $\sum_i \sum_{j \text{ is assigned to } i} p_{ij}$, further illustrating the problem of different algorithmic goals. Thus the VCG scheme cannot be used, and new methods must be developed.

Results for this problem domain depend on the specific assumptions about the structure of the processing time vectors. In the *related machines* case, $p_{ij} = p_j/s_i$ for any ij, where the p_j's are public knowledge, and the only secret parameter of player i is its *speed*, s_i.

Theorem 2 ([3,22]) *For job scheduling on related machines, there exists a truthful exponential-time mechanism that obtains the optimal makespan, and a truthful polynomial-time mechanism that obtains a 3-approximation to the optimal makespan.*

More details on this result are given in the entry on Mechanism for One Parameter Agents Single Buyer. The bottom line conclusion is that, although the social goal is different than welfare maximization, there still exists a truthful mechanism for this goal. A non-trivial approximation guarantee is achieved, even under the additional requirement of computational efficiency. However, this guarantee does not match the best possible without the truthfulness requirement, since in this case a PTAS is known.

Open Question 1 *Is there a truthful PTAS for makespan minimization in related machines?*

If the number of machines is fixed then [2] give such a truthful PTAS.

The above picture completely changes in the move to the more general case of *unrelated machines*, where the p_{ij}'s are allowed to be arbitrary:

Theorem 3 ([13,30]) *Any truthful scheduling mechanism for unrelated machines cannot approximate the optimal makespan by a factor better than $1 + \sqrt{2}$ (for deterministic mechanisms) and $2 - 1/m$ (for randomized mechanisms).*

Note that this holds regardless of computational considerations. In this case, switching from welfare maximization to makespan minimization results in a strong impossibility. On the possibilities side, virtually nothing (!) is known. The VCG mechanism (which minimizes the total social cost) is an m-approximation of the optimal makespan [32], and, in fact, nothing better is currently known:

Open Question 2 *What is the best possible approximation for truthful makespan minimization in unrelated machines?*

What caused the switch from "mostly possibilities" to "mostly impossibilities"? Related machines is a single-dimensional domain (players hold only one secret number), for which truthfulness is characterized by a simple monotonicity condition, that leaves ample flexibility for algorithmic design. Unrelated machines, on the other hand, are a multi-dimensional domain, and the algorithmic conditions implied by truthfulness in such a case are harder to work with. It is still unclear whether these conditions imply real mathematical impossibilities, or perhaps just pose harder obstacles that can be in principle solved. One multi-dimensional scheduling domain for which possibility results are known is the case where $p_{ij} \in \{L_j, H_j\}$, where the "low" 's and "high" 's are fixed and known. This case generalizes the classic multi-dimensional model of restricted machines ($p_{ij} \in \{p_j, \infty\}$), and admits a truthful 3-approximation [27].

Problem Domain 2: Digital Goods and Revenue Maximization

In the E-commerce era, a new kind of "digital goods" have evolved: goods with no marginal production cost, or, in other words, goods with unlimited supply. One example is songs being sold on the Internet. There is a sunk cost of producing the song, but after that, additional electronic copies incur no additional cost. How should such items be sold? One possibility is to conduct an *auction*. An auction is a one-sided market, where a monopolistic entity (the auctioneer) wishes to sell one or more items to a set of buyers.

In this setting, each buyer has a privately known value for obtaining one copy of the good. Welfare maximization simply implies the allocation of one good to every buyer, but a more interesting question is the question of revenue maximization. How should the auctioneer design the auction in order to maximize his profit? Standard tools from the study of revenue-maximizing auctions[1] suggest to simply declare a price-per-buyer, determined by the probability distribution of the buyer's value, and make a take-it-or-leave-it offer. However, such a mechanism needs to know the underlying distribution. Algorithmic mechanism design suggests an alternative, worst-case result, in the spirit of CS-type models and analysis.

Suppose that the auctioneer is required to sell all items in the same price, as is the case for many "real-life" monopolists, and denote by $F(\vec{v})$ the maximal revenue from a fixed-price sale to bidders with values $\vec{v} = v_1, \ldots v_n$, *assuming that all values are known*. Reordering indexes so that $v_1 \geq v_2 \geq \cdots \geq v_n$, let $F(\vec{v}) = \max_i i \cdot v_i$. The problem is, of-course, that in fact *nothing* about the values is known. Therefore, a truthful auction that extracts the players' values is in place. Can such an auction obtain a profit that is a constant fraction of $F(\vec{v})$, for any $\vec{v}$ (i. e. in the worst case)? Unfortunately, the answer is provably no [17]. The proof makes use of situations where the entire profit comes from the highest bidder. Since there is no potential for competition among bidders, a truthful auction cannot force this single bidder to reveal her value.

Luckily, a small relaxation in the optimality criteria significantly helps. Specifically, denote by $F^{(2)}(\vec{v}) = \max_{i \geq 2} i \cdot v_i$ (i. e. the benchmark is the auction that sells to at least two buyers).

Theorem 4 ([17,20]) *There exists a truthful randomized auction that obtains an expected revenue of at least $F^{(2)}/3.25$, even in the worst-case. On the other hand, no truthful auction can approximate $F^{(2)}$ within a factor better than 2.42.*

Several interesting formats of distribution-free revenue-maximizing auctions have been considered in the literature. The common building block in all of them is the random partitioning of the set of buyers to random subsets, analyzing each set separately, and using the results on the other sets. Each auction utilizes a different analysis on the two subsets, which yields slightly different approximation guarantees. [1] describe an elegant method to derandomize these type of auctions, while losing another factor of 4 in the approximation. More details on this problem domain can be found in the entry on Competitive Auctions.

Problem Domain 3: Combinatorial Auctions

Combinatorial auctions (CAs) are a central model with theoretical importance and practical relevance. It generalizes many theoretical algorithmic settings, like job scheduling and network routing, and is evident in many real-life situations. This new model has various pure computational aspects, and, additionally, exhibits interesting

[1] This model was not explicitly studied in classic auction theory, but standard results from there can be easily adjusted to this setting.

game theoretic challenges. While each aspect is important on its own, obviously only the integration of the two provides an acceptable solution.

A combinatorial auction is a multi-item auction in which players are interested in *bundles* of items. Such a valuation structure can represent substitutabilities among items, complementarities among items, or a combination of both. More formally, m items (Ω) are to be allocated to n players. Players value subsets of items, and $v_i(S)$ denotes i's value of a bundle $S \subseteq \Omega$. Valuations additionally satisfy: (i) monotonicity, i.e $v_i(S) \leq v_i(T)$ for $S \subseteq T$, and (ii) normalization, i. e. $v_i(\emptyset) = 0$. The literature has mostly considered the goal of maximizing the social welfare: find an allocation $(S_1, \ldots, S_n)$ that maximizes $\sum_i v_i(S_i)$.

Since a general valuation has size exponential in n and m, the representation issue must be taken into account. Two models are usually considered (see [11] for more details). In the *bidding languages* model, the bid of a player represents his valuation is a concise way. For this model it is NP-hard to approximate the social welfare within a ratio of $\Omega(m^{1/2-\epsilon})$, for any $\epsilon > 0$ (if "single-minded" bids are allowed; the exact definition is given below). In the *query access* model, the mechanism iteratively queries the players in the course of computation. For this model, any algorithm with polynomial communication cannot obtain an approximation ratio of $\Omega(m^{1/2-\epsilon})$ for any $\epsilon > 0$. These bounds are tight, as there exist a deterministic $\sqrt{m}$-approximation with polynomial computation and communication. Thus, for the general valuation structure, the computational status by itself is well-understood.

The basic incentives issue is again well-understood: VCG obtains truthfulness. Since VCG requires the exact optimum, which is NP-hard to compute, the two considerations therefore clash, when attempting to use classic techniques. Algorithmic mechanism design aims to develop new techniques, to integrate these two desirable aspects.

The first positive result for this integration challenge was given by [29], for the special case of "single-minded bidders": each bidder, i, is interested in a specific bundle S_i, for a value v_i (any bundle that contains S_i is worth v_i, and other bundles have zero value). Both v_i, S_i are private to the player i.

Theorem 5 ([29]) *There exists a truthful and polynomial-time deterministic combinatorial auction for single-minded bidders, which obtains a $\sqrt{m}$-approximation to the optimal social welfare.*

A possible generalization of the basic model is to assume that each item has B copies, and each player still desires at most one copy from each item. This is termed "multi-unit CA". As B grows, the integrality constraint of the problem reduces, and so one could hope for better solutions. Indeed, the next result exploits this idea:

Theorem 6 ([7]) *There exists a truthful and polynomial-time deterministic multi-unit CA, for $B \geq 3$ copies of each item, that obtains $O(B \cdot m^{1/(B-2)})$-approximation to the optimal social welfare.*

This auction copes with the representation issue (since general valuations are assumed) by accessing the valuations through a "demand oracle": given per-item prices $\{p_x\}_{x\in\Omega}$, specify a bundle S that maximizes $v_i(S) - \sum_{x\in S} p_x$.

Two main drawbacks of this auction motivate further research on the issue. First, as B gets larger it is reasonable to expect the approximation to approach 1 (indeed polynomial-time algorithms with such an approximation guarantee do exist). However here the approximation ratio does not decrease below $O(\log m)$ (this ratio is achieved for $B = O(\log m)$). Second, this auction does not provide a solution to the original setting, where $B = 1$, and, in general for small B's the approximation factor is rather high. One way to cope with these problems is to introduce randomness:

Theorem 7 ([26]) *There exists a truthful-in-expectation and polynomial-time randomized multi-unit CA, for any $B \geq 1$ copies of each item, that obtains $O(m^{1/(B+1)})$-approximation to the optimal social welfare.*

Thus, by allowing randomness, the gap from the standard computational status is being completely closed. The definition of truthfulness-in-expectation is the natural extension of truthfulness to a randomized environment: the *expected* utility of a player is maximized by being truthful.

However, this notion is strictly weaker than the deterministic notion, as this implicitly implies that players care only about the expectation of their utility (and not, for example, about the variance). This is termed "the risk-neutrality" assumption in the economics literature. An intermediate notion for randomized mechanisms is that of "universal truthfulness": the mechanism is truthful given any fixed result of the coin toss. Here, risk-neutrality is no longer needed. [15] give a universally truthful CA for $B = 1$ that obtains an $O(\sqrt{m})$-approximation. Universally truthful mechanisms are still weaker than deterministic truthful mechanisms, due to two reasons: (i) It is not clear how to actually create the correct and exact probability distribution with a deterministic computer. The situation here is different than in "regular" algorithmic settings, where various derandomization techniques can be employed, since these in general does not carry through the truthfulness property. (ii) Even if a natural randomness source exists, one cannot improve the quality of the actual output by repeating the computation several times (using the the law of large numbers). Such a repetition will again destroy truthfulness. Thus, exactly because the game-theoretic issues are being considered in parallel to the computational ones, the importance of determinism increases.

Open Question 3 *What is the best-possible approximation ratio that deterministic and truthful combinatorial auctions can obtain, in polynomial-time?*

There are many valuation classes, that restrict the possible valuations to some reasonable format (see [28] for more details). For example, sub-additive valuations are such that, for any two bundles $S, T, \subseteq \Omega$, $v(S \cup T) \leq v(S) + v(T)$. Such classes exhibit much better approximation guarantees, e. g. for sub-additive valuation a polynomial-time 2-approximation is known [16]. However, no polynomial-time truthful mechanism (be it randomized, or deterministic) with a constant approximation ratio, is known for any of these classes.

Open Question 4 *Does there exist polynomial-time truthful constant-factor approximations for special cases of CAs that are NP-hard?*

Revenue maximization in CAs is of-course another important goal. This topic is still mostly unexplored, with few exceptions. The mechanism [7] obtains the same guarantees with respect to the optimal revenue. Improved approximations exist for multi-unit auctions (where all items are identical) with budget constrained players [12], and for unlimited-supply CAs with single-minded bidders [6].

The topic of Combinatorial Auctions is discussed also in the entry on Multiple Item Auctions.

Problem Domain 4: Online Auctions

In the classic CS setting of "online computation", the input to an algorithm is not revealed all at once, before the computation begins, but gradually, over time (for a detailed discussion see the many entries on online problems in this book). This structure suits the auction world, especially in the new electronic environments. What happens when players arrive over time, and the auctioneer must make decisions facing only a subset of the players at any given time?

The integration of online settings, worst-case analysis, and auction theory, was suggested by [24]. They considered the case where players arrive one at a time, and the auctioneer must provide an answer to each player *as it arrives*, without knowing the future bids. There are k identical items, and each bidder may have a distinct value for every possible quantity of the item. These values are assumed to be marginally decreasing, where each marginal value lies in the interval $[\underline{v}, \bar{v}]$. The private information of a bidder includes both her valuation function, and her arrival time, and so a truthful auction need to incentivize the players to arrive on time (and not later on), and to reveal their true values. The most interesting result in this setting is for a large k, so that in fact there is a continuum of items:

Theorem 8 ([24]) *There exists a truthful online auction that simultaneously approximates, within a factor of $O(\log(\bar{v}/\underline{v}))$, the optimal offline welfare, and the offline revenue of VCG. Furthermore, no truthful online auction can obtain a better approximation ratio to either one of these criteria (separately).*

This auction has the interesting property of being a "posted price" auction. Each bidder is not required to reveal his valuation function, but, rather, he is given a price for each possible quantity, and then simply reports the desired quantity under these prices.

Ideas from this construction were later used by [10] to construct *two*-sided online auction markets, where multiple sellers and buyers arrive online.

This approximation ratio can be dramatically improved, to be a constant, 4, if one assumes that (i) there is only one item, and (ii) player values are i.i.d from some fixed distribution. No a–priori knowledge of this distribution is needed, as neither the mechanism nor the players are required to make any use of it. This work, [19], analyzes this by making an interesting connection to the class of "secretary problems".

A general method to convert online algorithms to online mechanisms is given by [4]. This is done for one item auctions, and, more generally, for one parameter domains. This method is competitive both with respect to the welfare and the revenue.

The revenue that the online auction of Theorem 8 manages to raise is competitive only with respect to VCG's revenue, which may be far from optimal. A parallel line of works is concerned with revenue maximizing auctions. To achieve good results, two assumptions need to be made: (i) there exists an unlimited supply of items (and recall from Sect. "Problem Domain 2: Digital Goods and Revenue Maximization" that $F(v)$ is the offline optimal monopolistic fixed-price revenue), and (ii) players cannot lie about their arrival time, only about their value. This last assumption is very strong, but apparently needed. Such auctions are termed here "value-truthful", indicating that "time-truthfulness" is missing.

Theorem 9 ([9]) *For any $\epsilon > 0$, there exists a value-truthful online auction, for the unlimited supply case, with expected revenue of at least $(F(v))/(1+\epsilon) - O(h/\epsilon^2)$.*

The construction exploits principles from learning theory in an elegant way. Posted price auctions for this case are also possible, in which case the additive loss increases to $O(h \log\log h)$. [19] consider fully-truthful online auctions for revenue maximization, but manage to obtain only very high (although fixed) competitive ratios. Constructing fully-truthful online auctions with a close-to-optimal revenue remains an open question. Another interesting open question involves multi-dimensional valuations. The work [24] remains the only work for players that may demand multiple items. However their competitive guarantees are quite high, and achieving better approximation guarantees (especially with respect to the revenue) is a challenging task.

Advanced Issues

Monotonicity What is the general way for designing a truthful mechanism? The straight-forward way is to check, for a given social choice function f, whether truthful prices exist. If not, try to "fix" f. It turns out, however, that there exists a more structured way, an *algorithmic* condition that will imply the *existence* of truthful prices. Such a condition shifts the designer back to the familiar territory of algorithmic design. Luckily, such a condition do exist, and is best described in the abstract social choice setting of Sect. "Problem Definition":

Definition 3 ([8,23]) A social choice function $f\colon V \rightarrow A$ is "weakly monotone" (W-MON) if for any i, $v_{-i} \in V_{-i}$, and any $v_i, v'_i \in V_i$, the following holds. Suppose that $f(v_i, v_{-i}) = a$, and $f(v'_i, v_{-i}) = b$. Then $v'_i(b) - v_i(b) \geq v'_i(a) - v_i(a)$.

In words, this condition states the following. Suppose that player i changes her declaration from v_i to v'_i, and this causes the social choice to change from a to b. Then it must be the case that i's value for b has increased in the transition from v_i to v'_i no-less than i's value for a.

Theorem 10 ([35]) *Fix a social choice function $f\colon V \rightarrow A$, where V is convex, and A is finite. Then there exist prices p such that $M = (f, p)$ is truthful if and only if f is weakly monotone.*

Furthermore, given a weakly monotone f, there exists an explicit way to determine the appropriate prices p (see [18] for details).

Thus, the designer should aim for weakly monotone algorithms, and need not worry about actual prices. But how difficult is this? For single-dimensional domains, it turns out that W-MON leaves ample flexibility for the algorithm designer. Consider for example the case where every alternative has a value of either 0 (the player "loses") or some $v_i \in \Re$ (the player "wins" and obtains a value v_i). In such a case, it is not hard to show that W-MON reduces to the following monotonicity condition: if a player wins with v_i, and increases her value to $v'_i > v_i$ (while v_{-i} remains fixed), then she must win with v'_i as well. Furthermore, in such a case, the price of a winning player must be set to the infimum over all winning values.

Impossibilities of truthful design It is fairly simple to construct algorithms that satisfy W-MON for single-dimensional domains, and a variety of positive results were obtained for such domains, in classic mechanism design, as well as in algorithmic mechanism design. But how hard is it to satisfy W-MON for multi-dimensional domains? This question is yet unclear, and seems to be one of the challenges of algorithmic mechanism design. The contrast between single-dimensionality and multi-dimensionality appears in all problem domains that were surveyed here, and seems to reflect some inherent difficulty that is not exactly understood yet. Given a social choice function f, call f *implementable* (in dominant strategies) if there exist prices p such that $M = (f, p)$ is truthful. The basic question is then *what forms of social choice functions are implementable.*

As detailed in the beginning, the welfare maximizing social choice function is implementable. This specific function can be slightly generalized to allow weights, in the following way: fix some non-negative real constants $\{w_i\}_{i=1}^n$ (not all are zero) and $\{\gamma_a\}_{a \in A}$, and choose an alternative that maximizes the *weighted* social welfare, i. e. $f(v) \in \operatorname{argmax}_{a \in A} \sum_i w_i v_i(a) + \gamma_a$. This class of functions is sometimes termed "affine maximizers". It turns out that these functions are also implementable, with prices similar in spirit to VCG. In the context of the above characterization question, one sharp result stands out:

Theorem 11 ([34]) *Fix a social choice function $f\colon V \rightarrow A$, such that (i) A is finite, $|A| \geq 3$, and f is onto A, and (ii) $V_i = \Re^A$ for all i. Then f is implementable (in dominant strategies) if and only if it is an affine maximizer.*

The domain V that satisfies $V_i = \Re^A$ for all i is term an "unrestricted domain". The theorem states that, if the domain is unrestricted, at least three alternatives are chosen, and the set A of alternatives is finite, then nothing besides affine maximizers can be implemented!

However, the assumption that the domain is unrestricted is very restrictive. All the above example domains exhibit some basic combinatorial structure, and are therefore restricted in some way. And as discussed above, for many restricted domains the theorem is simply not true. So what *is* the possibilities – impossibilities border? As mentioned above, this is an unsolved challenge. Lavi, Mu'alem, and Nisan [23] explore this question for Combinatorial Auctions and similar restricted domains, and reach partial answers. For example:

Theorem 12 **([23])** *Any truthful combinatorial auction or multi-unit auction among two players, that must always allocate all items, and that approximates the welfare by a factor better than 2, must be an affine maximizer.*

Of-course, this is far from being a complete answer. What happens if there are more than two players? And what happens if it is possible to "throw away" part of the items? These questions, and the more general and abstract characterization question, are all still open.

Alternative solution concepts In light of the conclusions of the previous section, a natural thought would be to re-examine the *solution concept* that is being used. Truthfulness relies on the strong concept of dominant strategies: for each player there is a unique strategy that maximizes her utility, no matter what the other players are doing. This is very strong, but it fits very well the worst-case way of thinking in CS. What other solution concepts can be used? As described above, randomization, and truthfulness-in-expectation, can help. A related concept, again for randomized mechanisms, is truthfulness with high probability. Another direction is to consider mechanisms where players cannot improve their utility *too much* by deviating from the truth-telling strategy [21].

Algorithm designers do not care so much about actually reaching an equilibrium point, or finding out what will the players play – the major concern is to guarantee the optimality of the solution, taking into account the strategic behavior of the players. Indeed, one way of doing this is to guarantee a good equilibrium point. But there is no reason to rule out mechanisms where several acceptable strategic choices for the players exist, provided that the approximation will be achieved *in each of these choices*.

As a first attempt, one is tempted to simply let the players try and improve the basic result by allowing them to lie. However, this can cause unexpected dynamics, as each player chooses her lies under some assumptions about the lies of the others, etc. etc. To avoid such an unpredictable situation, it is important to insist on using rigorous game theoretic reasoning to explain exactly why the outcome will be satisfactory.

The work [31] suggests the notion of "feasibly dominant" strategies, where players reveal the possible lies they consider, and the mechanism takes this into account. By assuming that the players are computationally bounded, one can show that, instead of actually "lying", the players will prefer to reveal their true types plus all the lies they might consider. In such a case, since the mechanism has obtained the true types of the players, a close-to-optimal outcome will be guaranteed.

Another definition tries to capture the initial intuition by using the classic game-theoretic notion of undominated strategies:

Definition 4 **([5])** A mechanism M is an "algorithmic implementation of a c-approximation (in undominated strategies)" if there exists a set of strategies, D, such that (i) M obtains a c-approximation for any combination of strategies from D, in polynomial time, and (ii) For any strategy not in D, there exists a strategy in D that weakly dominates it, and this transition is polynomial-time computable.

By the second condition, it is reasonable to assume that a player will indeed play *some* strategy in D, and, by the first condition, it does not matter what tuple of strategies in D will actually be chosen, as any of these will provide the approximation. This transfers some of the burden from the game-theoretic design to the algorithmic design, since now a guarantee on the approximation should bu provided for a larger range of strategies. [5] exploit this notion to design a deterministic CA for multi-dimensional players that achieves a close-to-optimal approximation guarantee. A similar-in-spirit notion, although a weaker one, is the notion of "Set-Nash" [25].

Applications

One of the popular examples to a "real-life" combinatorial auction is the spectrum auction that the US government conducts, in order to sell spectrum licenses. Typical bids reflect values for different spectrum ranges, to accommodate different geographical and physical needs, where different spectrum ranges may complement or substitute one another. The US government invests research efforts in order to determine the best format for such an auction, and auction theory is heavily exploited. Interestingly, the US law guides the authorities to allocate these spectrum ranges in a way that will maximize *the social welfare*, thus providing a good example for the usefulness of this goal.

Adword auctions are another new and fast-growing application of auction theory in general, and of the new algorithmic auctions in particular. These are auctions that determine the advertisements that web-search engines place close to the search results they show, after the user submits her search keywords. The interested companies compete, for every given keyword, on the right to place their ad on the results' page, and this turns out to be the main source of income for companies like Google. Several entries in this book touch on this topic in more details, including the entries on Adwords Pricing and on Position Auctions.

A third example to a possible application, in the meanwhile implemented only in the academic research labs, is the application of algorithmic mechanism design to pricing and congestion control in communication networks. The existing fixed pricing scheme has many disadvantages, both with respect to the needs of efficiently allocating the available resources, and with respect to the new opportunities of the Internet companies to raise more revenue due to specific types of traffic. Theory suggests solutions to both of these problems.

Cross References

► Adwords Pricing
► Competitive Auction
► False-Name-Proof Auction
► Generalized Vickrey Auction
► Incentive Compatible Selection
► Position Auction
► Truthful Multicast

Recommended Reading

The topics presented here are detailed in the textbook [33]. Section "Problem Definition" is based on the paper [32], that also coined the term "algorithmic mechanism design". The book [14] covers the various aspects of combinatorial auctions.

1. Aggarwal, G., Fiat, A., Goldberg, A., Immorlica, N., Sudan, M.: Derandomization of auctions. In: Proc. of the 37th ACM Symposium on Theory of Computing (STOC'05), 2005
2. Andelman, N., Azar, Y., Sorani, M.: Truthful approximation mechanisms for scheduling selfish related machines. In: Proc. of the 22nd International Symposium on Theoretical Aspects of Computer Science (STACS), 2005, pp. 69–82
3. Archer, A., Tardos, É.: Truthful mechanisms for one-parameter agents. In: Proc. 42nd Annual Symposium on Foundations of Computer Science (FOCS), 2001, pp. 482–491
4. Awerbuch, B., Azar, Y., Meyerson, A.: Reducing truth-telling online mechanisms to online optimization. In: Proc. of the 35th ACM Symposium on Theory of Computing (STOC'03), 2003
5. Babaioff, M., Lavi, R., Pavlov, E.: Single-value combinatorial auctions and implementation in undominated strategies. In: Proc. of the 17th Symposium on Discrete Algorithms (SODA), 2006
6. Balcan, M., Blum, A., Hartline, J., Mansour, Y.: Mechanism design via machine learning. In: Proc. of the 46th Annual Symposium on Foundations of Computer Science (FOCS'05), 2005
7. Bartal, Y., Gonen, R., Nisan, N.: Incentive compatible multiunit combinatorial auctions. In: Proc. of the 9th Conference on Theoretical Aspects of Rationality and Knowledge (TARK'03), 2003
8. Bikhchandani, S., Chatterjee, S., Lavi, R., Mu'alem, A., Nisan, N., Sen, A.: Weak monotonicity characterizes deterministic dominant-strategy implementation. Econometrica **74**, 1109–1132 (2006)
9. Blum, A., Hartline, J.: Near-optimal online auctions. In: Proc. of the 16th Symposium on Discrete Algorithms (SODA), 2005
10. Blum, A., Sandholm, T., Zinkevich, M.: Online algorithms for market clearing. J. ACM **53**(5), 845–879 (2006)
11. Blumrosen, L., Nisan, N.: On the computational power of iterative auctions. In: Proc. of the 7th ACM Conference on Electronic Commerce (EC'05), 2005
12. Borgs, C., Chayes, J., Immorlica, N., Mahdian, M., Saberi, A.: Multi-unit auctions with budget-constrained bidders. In: Proc. of the 7th ACM Conference on Electronic Commerce (EC'05), 2005
13. Christodoulou, G., Koutsoupias, E., Vidali, A.: A lower bound for scheduling mechanisms. In: Proc. 18th Symposium on Discrete Algorithms (SODA), 2007
14. Cramton, P., Shoham, Y., Steinberg, R.: Combinatorial Auctions. MIT Press (2005)
15. Dobzinski, S., Nisan, N., Schapira, M.: Truthful randomized mechanisms for combinatorial auctions. In: Proc. of the 38th ACM Symposium on Theory of Computing (STOC'06), 2006
16. Feige, U.: On maximizing welfare when utility functions are subadditive. In: Proc. of the 38th ACM Symposium on Theory of Computing (STOC'06), 2006
17. Goldberg, A., Hartline, J., Karlin, A., Saks, M., Wright, A.: Competitive auctions. Games Econ. Behav. **55**(2), 242–269 (2006)
18. Gui, H., Muller, R., Vohra, R.V.: Characterizing dominant strategy mechanisms with multi-dimensional types (2004). Working paper
19. Hajiaghayi, M., Kleinberg, R., Parkes, D.: Adaptive limited-supply online auctions. In: Proc. of the 6th ACM Conference on Electronic Commerce (EC'04), 2004
20. Hartline, J., McGrew, R.: From optimal limited to unlimited supply auctions. In: Proc. of the 7th ACM Conference on Electronic Commerce (EC'05), 2005
21. Kothari, A., Parkes, D., Suri, S.: Approximately-strategyproof and tractable multi-unit auctions. Decis. Support Syst. **39**, 105–121 (2005)
22. Kovács, A.: Fast monotone 3-approximation algorithm for scheduling related machines. In: Proc. 13th Annual European Symposium on Algorithms (ESA), 2005, pp. 616–627
23. Lavi, R., Mu'alem, A., Nisan, N.: Towards a characterization of truthful combinatorial auctions. In: Proc. of the 44rd Annual Symposium on Foundations of Computer Science (FOCS'03), 2003
24. Lavi, R., Nisan, N.: Competitive analysis of incentive compatible on-line auctions. Theor. Comput. Sci. **310**, 159–180 (2004)
25. Lavi, R., Nisan, N.: Online ascending auctions for gradually expiring items. In: Proc. of the 16th Symposium on Discrete Algorithms (SODA), 2005
26. Lavi, R., Swamy, C.: Truthful and near-optimal mechanism design via linear programming. In: Proc. 46th Annual Symposium

on Foundations of Computer Science (FOCS), 2005, pp. 595–604

27. Lavi, R., Swamy, C.: Truthful mechanism design for multi-dimensional scheduling via cycle monotonicity (2007). Working paper
28. Lehmann, B., Lehmann, D., Nisan, N.: Combinatorial auctions with decreasing marginal utilities. Games Econom. Behav. **55**(2), 270–296 (2006)
29. Lehmann, D., O'Callaghan, L., Shoham, Y.: Truth revelation in approximately efficient combinatorial auctions. J. ACM **49**(5), 577–602 (2002)
30. Mu'alem, A., Schapira, M.: Setting lower bounds on truthfulness. In: Proc. 18th Symposium on Discrete Algorithms (SODA), 2007
31. Nisan, N., Ronen, A.: Computationally feasible vcg mechanisms. In: Proc. of the 2nd ACM Conference on Electronic Commerce (EC'00), 2000
32. Nisan, N., Ronen, A.: Algorithmic mechanism design. Games Econom. Behav. **35**, 166–196 (2001)
33. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.: Algorithmic Game Theory. Cambridge University Press (2007). (expected to appear)
34. Roberts, K.: The characterization of implementable choice rules. In: Laffont, J.J. (ed.) Aggregation and Revelation of Preferences, pp. 321–349. North-Holland (1979)
35. Saks, M., Yu, L.: Weak monotonicity suffices for truthfulness on convex domains. In: Proc. 6th ACM Conference on Electronic Commerce (ACM-EC), 2005, pp. 286–293

Algorithms for Spanners in Weighted Graphs

2003; Baswana, Sen

Surender Baswana[1], Sandeep Sen[2]

[1] Department of Computer Science and Engineering,
IIT Kanpur, Kanpur, India

[2] Department of Computer Science and Engineering,
IIT Delhi, New Delhi, India

Keywords and Synonyms

Graph algorithms; Randomized algorithms; Shortest path; Spanner

Problem Definition

A spanner is a *sparse* subgraph of a given undirected graph that preserves approximate distance between each pair of vertices. More precisely, a t-spanner of a graph $G = (V, E)$ is a subgraph (V, E_S), $E_S \subseteq E$ such that, for any pair of vertices, their distance in the subgraph is at most t times their distance in the original graph, where t is called the *stretch factor*. The spanners were defined formally by Peleg and Schäffer [14], though the associated notion was used implicitly by Awerbuch [3] in the context of network synchronizers.

Computing a t-spanner of smallest size for a given graph is a well motivated combinatorial problem with many applications. However, computing t-spanner of smallest size for a graph is NP-hard. In fact, for $t > 2$, it is NP-hard [10] even to approximate the smallest size of a t-spanner of a graph with ratio $O(2^{(1-\mu)\ln n})$ for any $\mu > 0$. Having realized this fact, researchers have pursued another direction which is quite interesting and useful. Let S_G^t be the size of the sparsest t-spanner of a graph G, and let S_n^t be the maximum value of S_G^t over all possible graphs on n vertices. Does there exist a polynomial time algorithm which computes, for any weighted graph and parameter t, its t-spanner of size $O(S_n^t)$? Such an algorithm would be the best one can hope for given the hardness of the original t-spanner problem. Naturally the question arises as to how large can S_n^t be? A 43-year old girth lower bound conjecture by Erdös [12] implies that there are graphs on n vertices whose $2k$- as well as $(2k-1)$-spanner will require $\Omega(n^{1+1/k})$ edges. This conjecture has been proved for $k = 1, 2, 3$ and 5. Note that a $(2k-1)$-spanner is also a $2k$-spanner and the lower bound on the size is the same for both a $2k$-spanner and a $(2k-1)$-spanner. So the objective is to design an algorithm that, for any weighted graph on n vertices, computes a $(2k-1)$-spanner of $O(n^{1+1/k})$ size. Needless to say, one would like to design the fastest algorithm for this problem, and the most ambitious aim would be to achieve the linear time complexity.

Key Results

The key results of this article are two very simple algorithms which compute a $(2k-1)$-spanner of a given weighted graph $G = (V, E)$. Let n and m denote the number of vertices and edges of G, respectively. The first algorithm, due to Althöfer et al. [2], is based on a greedy strategy, and runs in $O(mn^{1+1/k})$ time. The second algorithm [6] is based on a very local approach and runs in the expected $O(km)$ time. To start with, consider the following simple observation. Suppose there is a subset $E_S \subset E$ that ensures the following proposition for every edge $(x, y) \in E \backslash E_S$.

> $\mathcal{P}_t(x, y)$: the vertices x and y are connected in the subgraph (V, E_S) by a path consisting of at most t edges, and the weight of each edge on this path is not more than that of the edge (x, y).

It follows easily that the sub graph (V, E_S) will be a t-spanner of G. The two algorithms for computing the $(2k-1)$-

spanner eventually compute the set E_S based on two completely different approaches.

Algorithm I

This algorithm selects edges for its spanner in a greedy fashion, and is similar to Kruskal's algorithm for computing a minimum spanning tree. The edges of the graph are processed in the increasing order of their weights. To begin with, the spanner $E_S = \emptyset$ and the algorithm adds edges to it gradually. The decision as to whether an edge, say (u, v), has to be added (or not) to E_S is made as follows:

If the distance between u and v in the subgraph induced by the current spanner edges E_S is more than $t \cdot \text{weight}(u, v)$, then add the edge (u, v) to E_S, otherwise discard the edge.

It follows that $\mathcal{P}_t(x, y)$ would hold for each edge of E missing in E_S, and so at the end, the subgraph (V, E_S) will be a t-spanner. A well known result in elementary graph theory states that a graph with more than $n^{1+1/k}$ edges must have a cycle of length at most $2k$. It follows from the above algorithm that the length of any cycle in the subgraph (V, E_S) has to be at least $t + 1$. Hence for $t = 2k - 1$, the number of edges in the subgraph (V, E_S) will be less than $n^{1+1/k}$. Thus Algorithm I computes a $(2k-1)$-spanner of size $O(n^{1+1/k})$, which is indeed optimal based on the lower bound mentioned earlier.

A simple $O(mn^{1+1/k})$ implementation of Algorithm I follows based on Dijkstra's algorithm. Cohen [9], and later Thorup and Zwick [18] designed algorithms for a $(2k-1)$-spanner with an improved running time of $O(kmn^{1/k})$. These algorithms rely on several calls to Dijkstra's single-source shortest-path algorithm for distance computation and therefore were far from achieving linear time. On the other hand, since a spanner must approximate all pairs distances in a graph, it appears difficult to compute a spanner by avoiding explicit distance information. Somewhat surprisingly, Algorithm II, described in the following section, avoids any sort of distance computation and achieves expected linear time.

Algorithm II

This algorithm employs a novel clustering based on a very local approach, and establishes the following result for the spanner problem.

> Given a weighted graph $G = (V, E)$, and an integer $k > 1$, a spanner of $(2k-1)$-stretch and $O(kn^{1+1/k})$ size can be computed in expected $O(km)$ time.

The algorithm executes in $O(k)$ rounds, and in each round it essentially explores adjacency list of each vertex to prune dispensable edges. As a testimony of its simplicity, we will present the entire algorithm for a 3-spanner and its analysis in the following section. The algorithm can be easily adapted in other computational models (parallel, external memory, distributed) with nearly optimal performance (see [6] for more details).

Computing a 3-Spanner in Linear Time To meet the size constraint of a 3-spanner, a vertex should contribute an average of $\sqrt{n}$ edges to the spanner. So the vertices with degree $O(\sqrt{n})$ are easy to handle since all their edges can be selected in the spanner. For vertices with higher degree a clustering (grouping) scheme is employed to tackle this problem which has its basis in *dominating sets*.

To begin with, there is a set of edges E' initialized to E, and an empty spanner E_S. The algorithm processes the edges E', moves some of them to the spanner E_S and discards the remaining ones. It does so in the following two phases.

1. *Forming the clusters*:
   A sample $\mathcal{R} \subset V$ is chosen by picking each vertex independently with probability $1/\sqrt{n}$. The clusters will be formed around these sampled vertices. Initially the clusters are $\{\{u\} | u \in \mathcal{R}\}$. Each $u \in \mathcal{R}$ is called the *center* of its cluster. Each unsampled vertex $v \in V - \mathcal{R}$ is processed as follows.
   (a) If v is not adjacent to any sampled vertex, then every edge incident on v is moved to E_S.
   (b) If v is adjacent to one or more sampled vertices, let $\mathcal{N}(v, \mathcal{R})$ be the sampled neighbor that is nearest[1] to v. The edge $(v, \mathcal{N}(v, \mathcal{R}))$ along with every edge that is incident on v with weight less than this edge is moved to E_S. The vertex v is added to the cluster centered at $\mathcal{N}(v, \mathcal{R})$.

   As a last step of the first phase, all those edges (u, v) from E' where u and v are not sampled and belong to the same cluster are discarded.

   Let V' be the set of vertices corresponding to the endpoints of the edges E' left after the first phase. It follows that each vertex from V' is either a sampled vertex or adjacent to some sampled vertex, and the step 1(b) has partitioned V' into disjoint clusters, each centered around some sampled vertex. Also note that, as a consequence of the last step, each edge of the set E' is an inter-cluster edge. The graph (V', E'), and the corresponding clustering of V' is passed onto the following (second) phase.
2. *Joining vertices with their neighboring clusters*:
   Each vertex v of graph (V', E') is processed as follows.

[1] Ties can be broken arbitrarily. However, it helps conceptually to assume that all weights are distinct.

Let $E'(v, c)$ be the edges from the set E' incident on v from a cluster c. For each cluster c that is a neighbor of v, the least-weight edge from $E'(v, c)$ is moved to E_S and the remaining edges are discarded.

The number of edges added to the spanner E_S during the algorithm described above can be bounded as follows. Note that the sample set $\mathcal{R}$ is formed by picking each vertex randomly and independently with probability $1/\sqrt{n}$. It thus follows from elementary probability that for each vertex $v \in V$, the expected number of incident edges with weights less than that of $(v, \mathcal{N}(v, \mathcal{R}))$ is at most $\sqrt{n}$. Thus the expected number of edges contributed to the spanner by each vertex in the first phase of the algorithm is at most $\sqrt{n}$. The number of edges added to the spanner in the second phase is $O(n|\mathcal{R}|)$. Since the expected size of the sample $\mathcal{R}$ is $\sqrt{n}$, therefore, the expected number of edges added to the spanner in the second phase is at most $n^{3/2}$. Hence the expected size of the spanner E_S at the end of Algorithm II as described above is at most $2n^{3/2}$. The algorithm is repeated if the size of the spanner exceeds $3n^{3/2}$. It follows using Markov's inequality that the expected number of such repetitions will be $O(1)$.

We now establish that E_S is a 3-spanner. Note that for every edge $(u, v) \notin E_S$, the vertices u and v belong to some cluster in the first phase. There are two cases now.

Case 1 : *(u and v belong to same cluster)*
Let u and v belong to the cluster centered at $x \in \mathcal{R}$. It follows from the first phase of the algorithm that there is a 2-edge path $u - x - v$ in the spanner with each edge not heavier than the edge (u, v). (This provides a justification for discarding all intra-cluster edges at the end of first phase).

Case 2 : *(u and v belong to different clusters)*
Clearly the edge (u, v) was removed from E' during phase 2, and suppose it was removed while processing the vertex u. Let v belong to the cluster centered at $x \in \mathcal{R}$.

In the beginning of the second phase let $(u, v') \in E'$ be the least weight edge among all the edges incident on u from the vertices of the cluster centered at x. So it must be that $\text{weight}(u, v') \leq \text{weight}(u, v)$. The processing of vertex u during the second phase of our algorithm ensures that the edge (u, v') gets added to E_S. Hence there is a path $\Pi_{uv} = u - v' - x - v$ between u and v in the spanner E_S, and its weight can be bounded as $\text{weight}(\Pi_{uv}) = \text{weight}(u, v') + \text{weight}(v', x) + \text{weight}(x, v)$. Since (v', x) and (v, x) were chosen in the first phase, it follows that $\text{weight}(v', x) \leq \text{weight}(u, v')$ and $\text{weight}(x, v) \leq \text{weight}(u, v)$. It follows that the spanner (V, E_S) has stretch 3. Moreover, both phases of the algorithm can be executed in $O(m)$ time using elementary data structures and bucket sorting.

The algorithm for computing a $(2k - 1)$-spanner executes k iterations where each iteration is similar to the first phase of the 3-spanner algorithm. For details and formal proofs, the reader may refer to [6].

Other Related Work

The notion of a spanner has been generalized in the past by many researchers.

Additive spanners: A t-spanner as defined above approximates pairwise distances with multiplicative error, and can be called a multiplicative spanner. In an analogous manner, one can define spanners that approximate pairwise distances with additive error. Such a spanner is called an additive spanner and the corresponding error is called a *surplus*. Aingworth et al. [1] presented the first additive spanner of size $O(n^{3/2} \log n)$ with surplus 2. Baswana et al. [7] presented a construction of $O(n^{4/3})$ size additive spanner with surplus 6. It is a major open problem if there exists any sparser additive spanner.

(α, β)-spanner: Elkin and Peleg [11] introduced the notion of an (α, β)-spanner for unweighted graphs, which can be viewed as a hybrid of multiplicative and additive spanners. An (α, β)-spanner is a subgraph such that the distance between any pair of vertices $u, v \in V$ in this subgraph is bounded by $\alpha\delta(u, v) + \beta$, where $\delta(u, v)$ is the distance between u and v in the original graph. Elkin and Peleg showed that an $(1 + \epsilon, \beta)$-spanner of size $O(\beta n^{1+\delta})$, for arbitrarily small $\epsilon, \delta > 0$, can be computed at the expense of a sufficiently large surplus β. Recently Thorup and Zwick [19] introduced a spanner where the additive error is sublinear in terms of the *distance* being approximated.

Other interesting variants of spanners include the *distance preserver* proposed by Bollobás et al. [8] and the *Light-weight* spanner proposed by Awerbuch et al. [4]. A subgraph is said to be a d-preserver if it preserves exact distances for each pair of vertices which are separated by distance at least d. A light-weight spanner tries to minimize the number of edges as well as the total edge weight. A *lightness* parameter is defined for a subgraph as the ratio of the total weight of all its edges and the weight of the minimum spanning tree of the graph. Awerbuch et al. [4] showed that for any weighted graph and integer $k > 1$, there exists a polynomially constructable $O(k)$-spanner with $O(k\rho n^{1+1/k})$ edges and $O(k\rho n^{1/k})$ lightness, where $\rho = \log(\text{Diameter})$.

In addition to the above work on the generalization of spanners, a lot of work has also been done on computing spanners for special classes of graphs, e. g., chordal graphs, unweighted graphs, and Euclidean graphs. For chordal graphs, Peleg and Schäffer [14] designed an algorithm that computes a 2-spanner of size $O(n^{3/2})$, and a 3-spanner of size $O(n \log n)$. For unweighted graphs Halperin and Zwick [13] gave an $O(m)$ time algorithm for this problem. Salowe [17] presented an algorithm for computing a $(1+\epsilon)$-spanner of a d-dimensional complete Euclidean graph in $O(n \log n + \frac{n}{\epsilon^d})$ time. However, none of the algorithms for these special classes of graphs seem to extend to general weighted undirected graphs.

Applications

Spanners are quite useful in various applications in the areas of distributed systems and communication networks. In these applications, spanners appear as the underlying graph structure. In order to build compact routing tables [16], many existing routing schemes use the edges of a sparse spanner for routing messages. In distributed systems, spanners play an important role in designing *synchronizers*. Awerbuch [3], and Peleg and Ullman [15] showed that the quality of a spanner (in terms of stretch factor and the number of spanner edges) is very closely related to the time and communication complexity of any synchronizer for the network. The spanners have also been used implicitly in a number of algorithms for computing all pairs of approximate shortest paths [5,9,18]. For a number of other applications, please refer to the papers [2,3,14,16].

Open Problems

The running time as well as the size of the $(2k-1)$-spanner computed by the Algorithm II described above are away from their respective worst case lower bounds by a factor of k. For any constant value of k, both these parameters are optimal. However, for the extreme value of k, that is, for $k = \log n$, there is a deviation by a factor of $\log n$. Is it possible to get rid of this multiplicative factor of k from the running time of the algorithm and/or the size of the $(2k-1)$-spanner computed? It seems that a more careful analysis coupled with advanced probabilistic tools might be useful in this direction.

Recommended Reading

1. Aingworth, D., Chekuri, C., Indyk, P., Motwani, R.: Fast estimation of diameter and shortest paths (without matrix multiplication). SIAM J. Comput. **28**, 1167–1181 (1999)
2. Althöfer, I., Das, G., Dobkin, D.P., Joseph, D., Soares J.: On sparse spanners of weighted graphs. Discret. Comput. Geom. **9**, 81–100 (1993)
3. Awerbuch, B.: Complexity of network synchronization. J. Assoc. Comput. Mach. **32**(4), 804–823 (1985)
4. Awerbuch, B., Baratz, A., Peleg, D.: Efficient broadcast and light weight spanners. Tech. Report CS92-22, Weizmann Institute of Science (1992)
5. Awerbuch, B., Berger, B., Cowen, L., Peleg D.: Near-linear time construction of sparse neighborhood covers. SIAM J. Comput. **28**, 263–277 (1998)
6. Baswana, S., Sen, S.: A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. Random Struct. Algorithms **30**, 532–563 (2007)
7. Baswana, S., Telikepalli, K., Mehlhorn, K., Pettie, S.: New construction of (α, β)-spanners and purely additive spanners. In: Proceedings of 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2005, pp. 672–681
8. Bollobás, B., Coppersmith, D., Elkin M.: Sparse distance preserves and additive spanners. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2003, pp. 414–423
9. Cohen, E.: Fast algorithms for constructing t-spanners and paths with stretch t. SIAM J. Comput. **28**, 210–236 (1998)
10. Elkin, M., Peleg, D.: Strong inapproximability of the basic k-spanner problem. In: Proc. of 27th International Colloquim on Automata, Languages and Programming, 2000, pp. 636–648
11. Elkin, M., Peleg, D.: $(1+\epsilon, \beta)$-spanner construction for general graphs. SIAM J. Comput. **33**, 608–631 (2004)
12. Erdös, P.: Extremal problems in graph theory. In: Theory of Graphs and its Applications (Proc. Sympos. Smolenice, 1963), pp. 29–36. Publ. House Czechoslovak Acad. Sci., Prague (1964)
13. Halperin, S., Zwick, U.: Linear time deterministic algorithm for computing spanners for unweighted graphs. unpublished manuscript (1996)
14. Peleg, D., Schäffer, A.A.: Graph spanners. J. Graph Theory **13**, 99–116 (1989)
15. Peleg, D., Ullman, J.D.: An optimal synchronizer for the hypercube. SIAM J. Comput. **18**, 740–747 (1989)
16. Peleg, D., Upfal, E.: A trade-off between space and efficiency for routing tables. J. Assoc. Comput Mach. **36**(3), 510–530 (1989)
17. Salowe, J.D.: Construction of multidimensional spanner graphs, with application to minimum spanning trees. In: ACM Symposium on Computational Geometry, 1991, pp. 256–261
18. Thorup, M., Zwick, U.: Approximate distance oracles. J. Assoc. Comput. Mach. **52**, 1–24 (2005)
19. Thorup, M., Zwick, U.: Spanners and emulators with sublinear distance errors. In: Proceedings of 17th Annual ACM-SIAM Symposium on Discrete Algorithms, 2006, pp. 802–809

All Pairs Shortest Paths in Sparse Graphs

2004; Pettie

SETH PETTIE
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA

Keywords and Synonyms

Shortest route; Quickest route

Problem Definition

Given a communications network or road network one of the most natural algorithmic questions is how to determine the shortest path from one point to another. The *all pairs* shortest path problem (APSP) is, given a directed graph $G = (V, E, \ell)$, to determine the distance and shortest path between every pair of vertices, where $|V| = n, |E| = m$, and $\ell : E \to \mathbb{R}$ is the edge length (or weight) function. The output is in the form of two $n \times n$ matrices: $D(u, v)$ is the distance from u to v and $S(u, v) = w$ if (u, w) is the first edge on a shortest path from u to v. The APSP problem is often contrasted with the *point-to-point* and *single source* (SSSP) shortest path problems. They ask for, respectively, the shortest path from a given source vertex to a given target vertex, and all shortest paths from a given source vertex.

Definition of Distance

If ℓ assigns only non-negative edge lengths then the definition of distance is clear: $D(u, v)$ is the length of the minimum length path from u to v, where the length of a path is the total length of its constituent edges. However, if ℓ can assign negative lengths then there are several sensible notations of distance that depend on how negative length cycles are handled. Suppose that a cycle C has negative length and that $u, v \in V$ are such that C is reachable from u and v reachable from C. Because C can be traversed an arbitrary number of times when traveling from u to v, there is no shortest path from u to v using a finite number of edges. It is sometimes assumed a priori that G has no negative length cycles; however it is cleaner to define $D(u, v) = -\infty$ if there is no finite shortest path. If $D(u, v)$ is defined to be the length of the shortest *simple* path (no repetition of vertices) then the problem becomes NP-hard.[1] One could also define distance to be the length of the shortest path without repetition of edges.

Classic Algorithms

The Bellman-Ford algorithm solves SSSP in $O(mn)$ time and under the assumption that edge lengths are non-negative, Dijkstra's algorithm solves it in $O(m + n \log n)$ time. There is a well known $O(mn)$-time shortest path preserving transformation that replaces any length function with a non-negative length function. Using this transformation and n runs of Dijkstra's algorithm gives an APSP algorithm running in $O(mn + n^2 \log n) = O(n^3)$ time. The Floyd–Warshall algorithm computes APSP in a more direct manner, in $O(n^3)$ time. Refer to [4] for a description of these algorithms. It is known that APSP on complete graphs is asymptotically equivalent to $(\min, +)$ matrix multiplication [1], which can be computed by a non-uniform algorithm that performs $O(n^{2.5})$ numerical operations [6].[2]

Integer-Weighted Graphs

Much recent work on shortest paths assume that edge lengths are integers in the range $\{-C, \ldots, C\}$ or $\{0, \ldots, C\}$. One line of research reduces APSP to a series of standard matrix multiplications. These algorithms are limited in their applicability because their running times scale linearly with C. There are faster SSSP algorithms for both non-negative edge lengths and arbitrary edge lengths. The former exploit the power of RAMs to sort in $o(n \log n)$ time and the latter are based on the *scaling* technique. See Zwick [19] for a survey of shortest path algorithms up to 2001.

Key Results

Pettie's APSP algorithm [13] adapts the *hierarchy* approach of Thorup [17] (designed for undirected, integer-weighted graphs) to general real-weighted directed graphs. Theorem 1 is the first improvement over the $O(mn + n^2 \log n)$ time bound of Dijkstra's algorithm on arbitrary real-weighted graphs.

Theorem 1 *Given a real-weighted directed graph, all pairs shortest paths can be solved in $O(mn + n^2 \log \log n)$ time.*

This algorithm achieves a logarithmic speedup through a trio of new techniques. The first is to exploit the necessary similarity between the SSSP trees emanating from nearby vertices. The second is a method for computing discrete approximate distances in real-weighted graphs. The third is a new hierarchy-type SSSP algorithm that runs in $O(m + n \log \log n)$ time when given suitably accurate approximate distances.
Theorem 1 should be contrasted with the time bounds of other hierarchy-type APSP algorithms [17,12,15].

Theorem 2 ([15], 2005) *Given a real-weighted undirected graph, APSP can be solved in $O(mn \log \alpha(m, n))$ time.*

Theorem 3 ([17], 1999) *Given an undirected graph $G(V, E, \ell)$, where ℓ assigns integer edge lengths in the range $\{-2^{w-1}, \ldots, 2^{w-1} - 1\}$, APSP can be solved in $O(mn)$ time on a RAM with w-bit word length.*

[1] If all edges have length -1 then $D(u, v) = -(n - 1)$ if and only if G contains a Hamiltonian path [7] from u to v.

[2] The fastest known $(\min, +)$ matrix multiplier runs n $O(n^3 (\log \log n)^3 / (\log n)^2)$ time [3].

Theorem 4 ([14], 2002) *Given a real-weighted directed graph, APSP can be solved in polynomial time by an algorithm that performs $O(mn \log \alpha(m, n))$ numerical operations, where α is the inverse-Ackermann function.*

A secondary result of [13,15] is that no *hierarchy-type* shortest path algorithm can improve on the $O(m + n \log n)$ running time of Dijkstra's algorithm.

Theorem 5 *Let G be an input graph such that the ratio of the maximum to minimum edge length is r. Any hierarchy-type SSSP algorithm performs $\Omega(m + \min\{n \log n, n \log r\})$ numerical operations if G is directed and $\Omega(m + \min\{n \log n, n \log\log r\})$ if G is undirected.*

Applications

Shortest paths appear as a subproblem in other graph optimization problems; the minimum weight perfect matching, minimum cost flow, and minimum mean-cycle problems are some examples. A well known commercial application of shortest path algorithms is finding efficient routes on road networks; see, for example, Google Maps, MapQuest, or Yahoo Maps.

Open Problems

The longest standing open shortest path problems are to improve the SSSP algorithms of Dijkstra's and Bellman-Ford on *real*-weighted graphs.

Problem 1 *Is there an $o(mn)$ time SSSP or point-to-point shortest path algorithm for arbitrarily weighted graphs?*

Problem 2 *Is there an $O(m) + o(n \log n)$ time SSSP algorithm for directed, non-negatively weighted graphs? For undirected graphs?*

A partial answer to Problem 2 appears in [15], which considers undirected graphs. Perhaps the most surprising open problem is whether there is any (asymptotic) difference between the complexities of the all pairs, single source, and point-to-point shortest path problems on arbitrarily weighted graphs.

Problem 3 *Is point-to-point shortest paths easier than all pairs shortest paths on arbitrarily weighted graphs?*

Problem 4 *Is there a genuinely subcubic APSP algorithm, i. e., one running in time $O(n^{3-\epsilon})$? Is there a subcubic APSP algorithm for integer-weighted graphs with weak dependence on the largest edge weight C, i. e., running in time $O(n^{3-\epsilon}\operatorname{polylog}(C))$?*

Experimental Results

See [9,16,5] for recent experiments on SSSP algorithms. On sparse graphs the best APSP algorithms use repeated application of an SSSP algorithm, possibly with some precomputation [16]. On dense graphs cache-efficiency becomes a major issue. See [18] for a cache conscious implementation of the Floyd–Warshall algorithm.

The trend in recent years is to construct a linear space data structure that can quickly answer exact or approximate point-to-point shortest path queries; see [10,6,2,11].

Data Sets

See [5] for a number of U.S. and European road networks.

URL to Code

See [8] and [5].

Cross References

► All Pairs Shortest Paths via Matrix Multiplication
► Single-Source Shortest Paths

Recommended Reading

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The design and analysis of computer algorithms. Addison-Wesley, Reading (1975)
2. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In transit to constant shortest-path queries in road networks. In: Proc. 9th Workshop on Algorithm Engineering and Experiments (ALENEX), 2007
3. Chan, T.: More algorithms for all-pairs shortest paths in weighted graphs. In: Proc. 39th ACM Symposium on Theory of Computing (STOC), 2007, pp. 590–598
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge (2001)
5. Demetrescu, C., Goldberg, A.V., Johnson, D.: 9th DIMACS Implementation challenge – shortest paths. http://www.dis.uniroma1.it/~challenge9/ (2006)
6. Fredman, M.L.: New bounds on the complexity of the shortest path problem. SIAM J. Comput. **5**(1), 83–89 (1976)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: a guide to NP-Completeness. Freeman, San Francisco (1979)
8. Goldberg, A.V.: AVG Lab. http://www.avglab.com/andrew/
9. Goldberg, A.V.: Shortest path algorithms: Engineering aspects. In: Proc. 12th Int'l Symp. on Algorithms and Computation (ISAAC). LNCS, vol. 2223, pp. 502–513. Springer, Berlin (2001)
10. Goldberg, A.V., Kaplan, H., Werneck, R.: Reach for A*: efficient point-to-point shortest path algorithms. In: Proc. 8th Workshop on Algorithm Engineering and Experiments (ALENEX), 2006
11. Knopp, S., Sanders, P., Schultes, D., Schulz, F., Wagner, D.: Computing many-to-many shortest paths using highway hierarchies. In: Proc. 9th Workshop on Algorithm Engineering and Experiments (ALENEX), 2007

12. Pettie, S.: On the comparison-addition complexity of all-pairs shortest paths. In: Proc. 13th Int'l Symp. on Algorithms and Computation (ISAAC), 2002, pp. 32–43
13. Pettie, S.: A new approach to all-pairs shortest paths on real-weighted graphs. Theor. Comput. Sci. **312**(1), 47–74 (2004)
14. Pettie, S., Ramachandran, V.: Minimizing randomness in minimum spanning tree, parallel connectivity and set maxima algorithms. In: Proc. 13th ACM-SIAM Symp. on Discrete Algorithms (SODA), 2002, pp. 713–722
15. Pettie, S., Ramachandran, V.: A shortest path algorithm for real-weighted undirected graphs. SIAM J. Comput. **34**(6), 1398–1431 (2005)
16. Pettie, S., Ramachandran, V., Sridhar, S.: Experimental evaluation of a new shortest path algorithm. In: Proc. 4th Workshop on Algorithm Engineering and Experiments (ALENEX), 2002, pp. 126–142
17. Thorup, M.: Undirected single-source shortest paths with positive integer weights in linear time. J. ACM **46**(3), 362–394 (1999)
18. Venkataraman, G., Sahni, S., Mukhopadhyaya, S.: A blocked all-pairs shortest paths algorithm. J. Exp. Algorithms **8** (2003)
19. Zwick, U.: Exact and approximate distances in graphs – a survey. In: Proc. 9th European Symposium on Algorithms (ESA), 2001, pp. 33–48. See updated version at http://www.cs.tau.ac.il/~zwick/

All Pairs Shortest Paths via Matrix Multiplication

2002; Zwick

Tadao Takaoka
Department of Computer Science
and Software Engineering, University of Canterbury,
Christchurch, New Zealand

Keywords and Synonyms

Shortest path problem; Algorithm analysis

Problem Definition

The all pairs shortest path (APSP) problem is to compute shortest paths between all pairs of vertices of a directed graph with non-negative real numbers as edge costs. Focus is given on shortest distances between vertices, as shortest paths can be obtained with a slight increase of cost. Classically, the APSP problem can be solved in cubic time of $O(n^3)$. The problem here is to achieve a sub-cubic time for a graph with small integer costs.

A directed graph is given by $G = (V, E)$, where $V = \{1, \dots, n\}$, the set of vertices, and E is the set of edges. The cost of edge $(i, j) \in E$ is denoted by d_{ij}. The (n, n)-matrix D is one whose (i,j) element is d_{ij}. It is assumed for simplicity that $d_{ij} > 0$ and $d_{ii} = 0$ for all $i \neq j$. If there is no edge from i to j, let $d_{ij} = \infty$. The cost, or distance, of a path is the sum of costs of the edges in the path. The length of a path is the number of edges in the path. The shortest distance from vertex i to vertex j is the minimum cost over all paths from i to j, denoted by d^*_{ij}. Let $D^* = \{d^*_{ij}\}$. The value of n is called the size of the matrices.

Let A and B be (n, n)-matrices. The three products are defined using the elements of A and B as follows: (1) Ordinary matrix product over a ring $C = AB$, (2) Boolean matrix product $C = A \cdot B$, and (3) Distance matrix product $C = A \times B$, where

$$(1)\ c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}\,, \quad (2)\ c_{ij} = \bigvee_{k=1}^{n} a_{ik} \wedge b_{kj}\,,$$

$$(3)\ c_{ij} = \min_{1 \le k \le n} \{a_{ik} + b_{kj}\}\,.$$

The matrix C is called a product in each case; the computational process is called multiplication, such as distance matrix multiplication. In those three cases, k changes through the entire set $\{1, \dots, n\}$. A partial matrix product of A and B is defined by taking k in a subset I of V. In other words, a partial product is obtained by multiplying a vertically rectangular matrix, $A(*, I)$, whose columns are extracted from A corresponding to the set I, and similarly a horizontally rectangular matrix, $B(I, *)$, extracted from B with rows corresponding to I. Intuitively I is the set of check points k, when going from i to j in the graph.

The best algorithm [3] computes (1) in $O(n^{\omega})$ time, where $\omega = 2.376$. Three decimal points are carried throughout this article. To compute (2), Boolean values 0 and 1 in A and B can be regarded as integers and use the algorithm for (1), and convert non-zero elements in the resulting matrix to 1. Therefore, this complexity is $O(n^{\omega})$. The witnesses of (2) are given in the witness matrix $W = \{w_{ij}\}$ where $w_{ij} = k$ for some k such that $a_{ik} \wedge b_{kj} = 1$. If there is no such k, $w_{ij} = 0$. The witness matrix $W = \{w_{ij}\}$ for (3) is defined by $w_{ij} = k$ that gives the minimum to c_{ij}. If there is an algorithm for (3) with $T(n)$ time, ignoring a polylog factor of n, the APSP problem can be solved in $\tilde{O}(T(n))$ time by the repeated squaring method, described as the repeated use of $D \leftarrow D \times D$ $O(\log n)$ times.

The definition here of computing shortest paths is to give a witness matrix of size n by which a shortest path from i to j can be given in $O(\ell)$ time where ℓ is the length of the path. More specifically, if $w_{ij} = k$ in the witness matrix $W = \{w_{ij}\}$, it means that the path from i to j goes through k. Therefore, a recursive function $path(i, j)$ is defined by

$(path(i,k), k, path(k,j))$ if $w_{ij} = k > 0$ and nil if $w_{ij} = 0$, where a path is defined by a list of vertices excluding endpoints. In the following sections, k is recorded in w_{ij} whenever k is found such that a path from i to j is modified or newly set up by paths from i to k and from k to j. Preceding results are introduced as a framework for the key results.

Alon–Galil–Margalit Algorithm

The algorithm by Alon, Galil, and Margalit [1] is reviewed. Let the costs of edges of the given graph be ones. Let $D^{(\ell)}$ be the ℓth approximate matrix for D^* defined by $d_{ij}^{(\ell)} = d_{ij}^*$ if $d_{ij}^* \le \ell$, and $d_{ij}^{(\ell)} = \infty$ otherwise. Let A be the adjacency matrix of G, that is, $a_{ij} = 1$ if there is an edge (i,j), and $a_{ij} = 0$ otherwise. Let $a_{ii} = 1$ for all i. The algorithm consists of two phases. In the first phase, $D^{(\ell)}$ is computed for $\ell = 1, \ldots, r$, by checking the (i,j)-element of $A^{\ell} = \{a_{ij}^{\ell}\}$. Note that if $a_{ij}^{\ell} = 1$, there is a path from i to j of length ℓ or less. Since Boolean mutrix multiplication can be computed in $O(n^{\omega})$ time, the computing time of this part is $O(rn^{\omega})$.

In the second phase, the algorithm computes $D^{(\ell)}$ for $\ell = r, \lceil 3/2\, r\rceil, \lceil 3/2 \lceil 3/2\, r\rceil\rceil, \cdots, n'$ by repeated squaring, where n' is the smallest integer in this sequence of ℓ such that $\ell \ge n$. Let $T_{i\alpha} = \{j | d_{ij}^{(\ell)} = \alpha\}$, and $I_i = T_{i\alpha}$ such that $|T_{i\alpha}|$ is minimum for $\lceil \ell/2 \rceil \le \alpha \le \ell$. The key observation in the second phase is that it is only needed to check k in I_i whose size is not larger than $2n/\ell$, since the correct distances between $\ell + 1$ and $\lceil 3\ell/2 \rceil$ can be obtained as the sum $d_{ik}^{(\ell)} + d_{kj}^{(\ell)}$ for some k satisfying $\lceil \ell/2 \rceil \le d_{ik}^{(\ell)} \le \ell$. The meaning of I_i is similar to I for partial products except that I varies for each i. Hence the computing time of one squaring is $O(n^3/\ell)$. Thus, the time of the second phase is given with $N = \lceil \log_{3/2} n/r \rceil$ by $O\big(\sum_{s=1}^{N} n^3/((3/2)^s r)\big) = O(n^3/r)$. Balancing the two phases with $rn^{\omega} = n^3/r$ yields $O(n^{(\omega+3)/2}) = O(n^{2.688})$ time for the algorithm with $r = O(n^{(3-\omega)/2})$.

Witnesses can be kept in the first phase in time polylog of n by the method in [2]. The maintenance of witnesses in the second phase is straightforward.

When a directed graph G whose edge costs are integers between 1 and M is given, where M is a positive integer, the graph G can be converted to G' by replacing each edge by up to M edges with unit cost. Obviously the problem for G can be solved by applying the above algorithm to G', which takes $O((Mn)^{(\omega+3)/2})$ time. This time is subcubic when $M < n^{0.116}$. The maintenance of witnesses has an extra polylog factor in each case.

For undirected graphs with unit edge costs, $\tilde{O}(n^{\omega})$ time is known in Seidel [7].

Takaoka algorithm

When the edge costs are bounded by a positive integer M, a better algorithm can be designed than in the above as shown in Takaoka [9]. Romani's algorithm [6] for distance matrix multiplication is reviewed briefly.

Let A and B be (n,m) and (m,n) distance matrices whose elements are bounded by M or infinite. Let the diagonal elements be 0. A and B are converted into A' and B' where $a'_{ij} = (m+1)^{M-a_{ij}}$, if $a_{ij} \neq \infty$, 0 otherwise, and $b'_{ij} = (m+1)^{M-b_{ij}}$, if $b_{ij} \neq \infty$, 0 otherwise.

Let $C' = A'B'$ be the product by ordinary matrix multiplication and $C = A \times B$ be that by distance matrix multiplication. Then it holds that

$$c'_{ij} = \sum_{k=1}^{m} (m+1)^{2M-(a_{ik}+b_{kj})}, \quad c_{ij} = 2M - \lfloor \log_{m+1} c'_{ij} \rfloor .$$

This distance mutrix multiplication is called (n,m)-Romani. In this section the above multiplication is used with square matrices, that is, (n,n)-Romani is used. In the next section, the case where $m < n$ is dealt with.

C can be computed with $O(n^{\omega})$ arithmetic operations on integers up to $(n+1)^M$. Since these values can be expressed by $O(M \log n)$ bits and Schönhage and Strassen's algorithm [8] for multiplying k-bit numbers takes $O(k \log k \log\log k)$ bit operations, C can be computed in $O(n^{\omega} M \log n \log(M \log n) \log\log(M \log n))$ time, or $\tilde{O}(Mn^{\omega})$ time.

The first phase is replaced by the one based on (n,n)-Romani, and modify the second phase based on path lengths, not distances.

Note that the bound M is replaced by ℓM in the distance matrix multiplication in the first phase. Ignoring polylog factors, the time for the first phase is given by $\tilde{O}(n^{\omega} r^2 M)$. It is assumed that M is $O(n^k)$ for some constant k. Balancing this complexity with that of the second phase, $O(n^3/r)$, yields the total computing time of $\tilde{O}(n^{(6+\omega)/3} M^{1/3})$ with the choice of $r = O(n^{(3-\omega)/3} M^{-1/3})$. The value of M can be almost $O(n^{0.624})$ to keep the complexity within sub-cubic.

Key Results

Zwick improved the Alon–Galil–Margalit algorithm in several ways. The most notable is an improvement of the time for the APSP problem with unit edge costs from $O(n^{2.688})$ to $O(n^{2.575})$. The main accelerating engine in Alon–Galil–Margalit [1] was the fast Boolean matrix multiplication and that in Takaoka [9] was the fast distance matrix multiplication by Romani, both powered by the fast matrix multiplication of square matrices.

In this section, the engine is the fast distance matrix multiplication by Romani powered by the fast matrix multiplication of rectangular matrices given by Coppersmith [4], and Huang and Pan [5]. Let $\omega(p, q, r)$ be the exponent of time complexity for multiplying (n^p, n^q) and (n^q, n^r) matrices. Suppose the product of (n, m) matrix and (m, n) matrix can be computed with $O(n^{\omega(1,\mu,1)})$ arithmetic operations, where $m = n^\mu$ with $0 \le \mu \le 1$. Several facts such as $O(n^{\omega(1,1,1)}) = O(n^{2.376})$ and $O(n^{\omega(1,0.294,1)}) = \tilde{O}(n^2)$ are known. To compute the product of (n, n) square matrices, $n^{1-\mu}$ matrix multiplications are needed, resulting in $O(n^{\omega(1,\mu,1)+1-\mu})$ time, which is reformulated as $O(n^{2+\mu})$, where μ satisfies the equation $\omega(1, \mu, 1) = 2\mu + 1$. Currently the best-known value for μ is $\mu = 0.575$, so the time becomes $O(n^{2.575})$, which is not as good as $O(n^{2.376})$. So the algorithm for rectangular matrices is used in the following.

The above algorithm is incorporated into (n, m)-Romani with $m = n^\mu$ and $M = n^t$ for some $t > 0$, and the computing time of $\tilde{O}(Mn^{\omega(1,\mu,1)})$. The next step is how to incorporate (n, m)-Romani into the APSP algorithm. The first algorithm is a mono-phase algorithm based on repeated squaring, similar to the second phase of the algorithm in [1]. To take advantage of rectangular matrices in (n, m)-Romani, the following definition of the bridging set is needed, which plays the role of the set I in the partial distance matrix product in Sect. "Problem Definition".

Let $\delta(i, j)$ be the shortest distance from i to j, and $\eta(i, j)$ be the minimum length of all shortest paths from i to j. A subset I of V is an ℓ-bridging set if it satisfies the condition that if $\eta(i, j) \ge \ell$, there exists $k \in I$ such that $\delta(i, j) = \delta(i, k) + \delta(k, j)$. I is a strong ℓ-bridging set if it satisfies the condition that if $\eta(i, j) \ge \ell$, there exists $k \in I$ such that $\delta(i, j) = \delta(i, k) + \delta(k, j)$ and $\eta(i, j) = \eta(i, k) + \eta(k, j)$. Note that those two sets are the same for a graph with unit edge costs.

Note that if $(2/3)\ell \le \mu(i, j) \le \ell$ and I is a strong $\ell/3$-bridging set, there is a $k \in I$ such that $\delta(i, j) = \delta(i, k) + \delta(k, j)$ and $\mu(i, j) = \mu(i, k) + \mu(k, j)$. With this property of strong bridging sets, (n, m)-Romani can be used for the APSP problem in the following way. By repeated squaring in a similar way to Alon–Galil–Margalit, the algorithm computes $D^{(\ell)}$ for $\ell = 1, \lceil 3/2 \rceil, \lceil 3/2 \lceil 3/2 \rceil \rceil, \ldots, n'$, where n' is the first value of ℓ that exceeds n, using various types of set I described below. To compute the bridging set, the algorithm maintains the witness matrix with extra polylog factor in the complexity. In [10], there are three ways for selecting the set I. Let $|I| = n^r$ for some r sucn that $0 \le r \le 1$.

(1) Select $9n \ln n/\ell$ vertices from V at random. In this case, it can be shown that the algorithm solves the APSP problem with high probability, i. e., with $1 - 1/n^c$ for some constant $c > 0$, which can be shown to be 3. In other words, I is a strong $\ell/3$-bridging set with high probability. The time T is dominated by (n, m)-Romani. It holds that $T = \tilde{O}(\ell M n^{\omega(1,r,1)})$, since the magnitude of matrix elements can be up to ℓM. Since $m = O(n \ln n/\ell) = n^r$, it holds that $\ell = \tilde{O}(n^{1-r})$, and thus $T = O(Mn^{1-r} n^{\omega(1,r,1)})$. When $M = 1$, this bound on r is $\mu = 0.575$, and thus $T = O(n^{2.575})$. When $M = n^t \ge 1$, the time becomes $O(n^{2+\mu(t)})$, where $t \le 3 - \omega = 0.624$ and $\mu = \mu(t)$ satisfies $\omega(1, \mu, 1) = 1 + 2\mu - t$. It is determined from the best known $\omega(1, \mu, 1)$ and the value of t. As the result is correct with high probability, this is a randomized algorithm.

(2) Consider the case of unit edge costs here. In (1), the computation of witnesses is an extra thing, i. e., not necessary if only shortest distances are needed. To achieve the same complexity in the sense of an exact algorithm, not a randomized one, the computation of witnesses is essential. As mentioned earlier, maintenance of witnesses, that is, matrix W, can be done with an extra polylog factor, meaning the analysis can be focused on Romani within the $\tilde{O}$-notation. Specifically I is selected as an $\ell/3$-bridging set, which is strong with unit edge costs. To compute I as an $O(\ell)$-bridging set, obtain the vertices on the shortest path from i to j for each i and j using the witness matrix W in $O(\ell)$ time. After obtaining those n^2 sets spending $O(\ell n^2)$ time, it is shown in [10] how to obtain a $O(\ell)$-bridging set of $O(n \ln n/\ell)$ size within the same time complexity. The process of obtaining the bridging set must stop at $\ell = n^{1/2}$ as the process is too expensive beyond this point, and thus the same bridging set is used beyond this point. The time before this point is the same as that in (1), and that after this point is $\tilde{O}(n^{2.5})$. Thus, this is a two-phase algorithm.

(3) When edge costs are positive and bounded by $M = n^t > 0$, a similar procedure can be used to compute an $O(\ell)$-bridging set of $O(n \ln n/\ell)$ size in $\tilde{O}(\ell n^2)$ time. Using the bridging set, the APSP problem can be solved in $\tilde{O}(n^{2+\mu(t)})$ time in a similar way to (1). The result can be generalized into the case where edge costs are between $-M$ and M within the same time complexity by modifying the procedure for computing an ℓ-bridging set, provided there is no negative cycle. The details are shown in [10].

Applications

The eccentricity of a vertex v of a graph is the greatest distance from v to any other vertices. The diameter of a graph is the greatest eccentricity of any vertices. In other words, the diameter is the greatest distance between any pair of vertices. If the corresponding APSP problem is solved, the maximum element of the resulting matrix is the diameter.

Open Problems

Two major challenges are stated here among others. The first is to improve the complexity of $\tilde{O}(n^{2.575})$ for the APSP with unit edge costs. The other is to improve the bound of $M < O(n^{0.624})$ for the complexity of the APSP with integer costs up to M to be sub-cubic.

Cross References

► All Pairs Shortest Paths in Sparse Graphs
► Fully Dynamic All Pairs Shortest Paths

Recommended Reading

1. Alon, N., Galil, Z., Margalit, O.: On the exponent of the all pairs shortest path problem. In: Proc. 32th IEEE FOCS, pp. 569–575. IEEE Computer Society, Los Alamitos, USA (1991). Also JCSS **54**, 255–262 (1997)
2. Alon, N., Galil, Z., Margalit, O., Naor, M.: Witnesses for Boolean matrix multiplication and for shortest paths. In: Proc. 33th IEEE FOCS, pp. 417–426. IEEE Computer Society, Los Alamitos, USA (1992)
3. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symb. Comput. **9**, 251–280 (1990)
4. Coppersmith, D.: Rectangular matrix multiplication revisited. J. Complex. **13**, 42–49 (1997)
5. Huang, X., Pan, V.Y.: Fast rectangular matrix multiplications and applications. J. Complex. **14**, 257–299 (1998)
6. Romani, F.: Shortest-path problem is not harder than matrix multiplications. Info. Proc. Lett. **11**, 134–136 (1980)
7. Seidel, R.: On the all-pairs-shortest-path problem. In: Proc. 24th ACM STOC pp. 745–749. Association for Computing Machinery, New York, USA (1992) Also JCSS **51**, 400–403 (1995)
8. Schönhage, A., Strassen, V.: Schnelle Multiplikation Großer Zahlen. Computing **7**, 281–292 (1971)
9. Takaoka, T.: Sub-cubic time algorithms for the all pairs shortest path problem. Algorithmica **20**, 309–318 (1998)
10. Zwick, U.: All pairs shortest paths using bridging sets and rectangular matrix multiplication. J. ACM **49**(3), 289–317 (2002)

Alternative Performance Measures in Online Algorithms

2000; Koutsoupias, Papadimitriou

Esteban Feuerstein
Department of Computing, University of Buenos Aires,
Buenos Aires, Argentina

Keywords and Synonyms

Diffuse adversary model for online algorithms; Comparative analysis for online algorithms

Problem Definition

Even if online algorithms had been studied for around thirty years, the explicit introduction of *competitive analysis* in the seminal papers by Sleator and Tarjan [8] and Manasse, McGeoch and Sleator [6] sparked an extraordinary boom in research about these class of problems and algorithms, so both concepts (online algorithms and competitive analysis) have been strongly related since. However, rather early in its development, some criticism arose regarding the realism and practicality of the model mainly because of its pessimism. That characteristic, in some cases, attempts on the ability of the model to distinguish, between good and bad algorithms. In a 1994 paper called *Beyond competitive analysis* [3], Koutsoupias and Papadimitriou proposed and explored two alternative performance measures for on-line algorithms, both very much related to competitive analysis and yet avoiding the weaknesses that caused the aforementioned criticism. The final version of that work appeared in 2000 [4].

In competitive analysis, the performance of an online algorithm is compared against an all-powerful adversary on a worst-case input. The competitive ratio of an algorithm A is defined as the worst possible ratio

$$R_A = \max_x \frac{A(x)}{opt(x)} ,$$

where x ranges over all possible inputs of the problem and $A(x)$ and $opt(x)$ are respectively the costs of the solutions obtained by algorithm A and the optimum offline algorithm for input x[1]. This notion can be extended to define the competitive ratio of a problem, as the minimum competitive ratio of an algorithm for it, namely

$$R = \min_A R_A = \min_A \max_x \frac{A(x)}{opt(x)} .$$

The main criticism to this approach has been that, with the characteristic pessimism common to all kinds of worst-case analysis, it fails to discriminate between algorithms that could have different performances under different conditions. Moreover, algorithms that "try" to perform well relative to this worst case measure many times fail to behave well in front of many "typical" inputs. This arguments can be more easily contested in the (rare) scenarios where the very strong assumption that *nothing* is known about the distribution of the input holds. But, this is rarely the case in practice.

[1] In this article all problems are assumed to be online minimization problems, therefore the objective is to minimize costs. All the results presented here are valid for online maximization problems with the proper adjustments to the definitions.

The paper by Koutsoupias and Papadimitriou proposes and studies two refinements of competitive analysis which try to overcome all these concerns. The first of them is the *diffuse adversary model*, which points at the cases where something is known about the input: its probabilistic distribution. With this in mind, the performance of an algorithm is evaluated comparing its expected cost with the one of an optimal algorithm for inputs following that distribution.

The second refinement is called *comparative analysis*. This refinement is based on the notion of *information regimes*. According to this, competitive analysis is interpreted as the comparison between two different information regimes, the online and the offline ones. But this vision entails that those information regimes are just particular, extreme cases of a large set of possibilities, among which, for example, the set of algorithms that know in advance some prefix of the awaiting input (finite lookahead algorithms).

Key Results

Diffuse Adversaries

The competitive ratio of an algorithm A against a class Δ of input distributions is the infimum c such that the algorithm is c-competitive when the input is restricted to that class. That happens whenever there exists a constant d such that, for all distributions $D \in \Delta$,

$$\mathcal{E}_D(A(x)) \leq c\mathcal{E}_D(opt(x)) + d\,,$$

where $\mathcal{E}_D$ stands for the mathematical expectation over inputs following distribution D. The competitive ratio $R(\Delta)$ of the class of distributions Δ is the minimum competitive ratio achievable by an online algorithm against Δ.

The model is applied to the traditional Paging problem, for the class of distributions Δ_ϵ. Δ_ϵ is the class that contains all probability distributions such that, given a request sequence and a page p, the probability that the next requested page is p is not more than ϵ. It is shown that the well-known online algorithm LRU achieves the optimal competitive ratio $R(\Delta_\epsilon)$ for all ϵ, that is, it is optimal against any adversary that uses a distribution in this class.

The proof of this result makes strong use of the *work function* concept introduced in [5], that is used as a tool to track the behavior of the optimal offline algorithm and estimate the optimal cost for a sequence of requests, and that of *conservative adversaries*, which are adversaries that assign higher probabilities to pages that have been requested more recently. This kind of adversary is consistent with *locality of reference*, a concept that has been always connected to Paging algorithms and competitive analysis (though in [1] another family of distributions is proposed, and analyzed within this framework, which better captures this notion).

The first result states that, for any adversary $D \in \Delta_\epsilon$, there is a conservative adversary $\hat{D} \in \Delta_\epsilon$ such that the competitive ratio of LRU against $\hat{D}$ is at least the competitive ratio of LRU against D. Then it is shown that for any conservative adversary $D \in \Delta_\epsilon$ against LRU, there is a conservative adversary $D' \in \Delta_\epsilon$, against an on-line algorithm A such that the competitive ratio of LRU against D is at most the competitive ratio of A against D'. In other words, for any ϵ, LRU has the optimal competitive ratio $R(\Delta_\epsilon)$ for the diffuse adversary model. This is the main result in the first part of [4].

The last remaining point refers to the value of the optimal competitive ratio of LRU for the Paging problem. As it is shown, that value is not easy to compute. For the extreme values of ϵ (the cases in which the adversary has complete and almost no control of the input, respectively), $R(\Delta_1) = k$, where k is the size of the cache, and also $\lim_{\epsilon \to 0} R(\Delta_\epsilon) = 1$. Later work by Young [9] allowed to estimate $R(\Delta_\epsilon)$ within (almost) a factor of two. For values of ε around the threshold $1/k$ the optimal ratio is $\Theta(\ln k)$, for values below that threshold the values tend rapidly to $O(1)$, and above it to $\Theta(k)$.

Comparative Analysis

Comparative analysis is a generalization of competitive analysis that allows to compare classes of algorithms, and not just individual algorithms. This new idea may be used to contrast the behaviors of algorithms obeying to arbitrary information regimes. In a few words, an information regime is a class of algorithms that acquire knowledge of the input in the same way, or at similar "rates", so both classes of online and offline algorithms are particular instances of this concept (the former know the input step by step, the latter receive all the information before having to produce any output).

The idea of comparative analysis is to measure the relative quality of two classes of algorithms by the maximum possible quotient of the results obtained by algorithms in each of the classes for the same input.

Formally, if $\mathcal{A}$ and $\mathcal{B}$ are classes of algorithms, the *comparative ratio* $R(\mathcal{A}, \mathcal{B})$ is defined as

$$R(\mathcal{A}, \mathcal{B}) = \max_{B \in \mathcal{B}} \min_{A \in \mathcal{A}} \max_{x} \frac{A(x)}{B(x)}\,.$$

With this definition, if $\mathcal{B}$ is the class of all algorithms, and $\mathcal{A}$ is the class of on-line algorithms, then the comparative ratio coincides with the competitive ratio.

The concept is illustrated determining how beneficial it can be to allow some *lookahead* to algorithms for *Metrical Task Systems (MTS)*. MTS are an abstract model that has been introduced in [2], and generalizes a wide family of on-line problems, among which Paging, the k-server problem, list accessing, and many other more. In a Metrical Task System a server can travel through the points of a Metric Space (states) while serving a sequence of requests or Tasks. The cost of serving a task depends on the state in which the server is, and the total cost for the sequence is given by the sum of the distance traveled plus the cost of servicing all the tasks. The meaning of the lookahead in this context is that the server can decide where to serve the next task based not only on the past movements and input but also on some fixed number of future requests.

The main result here (apart from the definition of the model itself) is that, for Metrical Task Systems, the Comparative Ratio for the class of online algorithms versus that of algorithms with lookahead l (respectively $\mathcal{L}_0$ and $\mathcal{L}_l$) is not more than $2l + 1$. That is, for this family of problems the benefit obtainable from lookahead is never more than two times the size of the lookahead plus one. The result is completed showing particular cases in which the equality holds.

Finally, for particular Metrical Task System the power of lookahead is shown to be strictly less than that: the last important result of this section shows that for the Paging Problem, the comparative ratio is exactly $min\{l + 1, k\}$, that is, the benefit of using lookahead l is the minimum between the size of the cache and the size of the lookahead window plus one.

Applications

As it is mentioned in the introduction of [4], the ideas presented therein are useful to have a better and more precise analysis of the performance of online algorithms. Also, the diffuse adversary model may prove useful to depict characteristics of the input that are probabilistic in nature (e. g. locality). An example in this direction is a paper by Becchetti [1], that uses a diffuse adversary with the intention of better modeling the locality of reference phenomenon that characterizes practical applications of Paging. In the distributions considered there the probability of requesting a page is also a function of the page's age, and it is shown that the competitive ratio of LRU becomes constant as locality increases.

A different approach is taken however in [7]. There the Paging problem with variable cache size is studied and it is shown that the approach of the expected competitive ratio in the diffuse adversary model can be misleading, while they propose the use of the *average performance ratio* instead.

Open Problems

It is an open problem to determine the exact competitive ratio against a diffuse adversary of known algorithms, for example FIFO, for the Paging problem. FIFO is known to be worse in practice than LRU, so proving that the former is suboptimal for some values of ε would give more support to the model.

An open direction presented in the paper is to consider what they call the *Markov diffuse adversary*, which as it is suggested by the name, refers to an adversary that generates the sequence of requests following a Markov process with output.

The last direction of research suggested is to use the idea of comparative analysis to compare the efficiency of agents or robots with different capabilities (for example with different vision ranges) to perform some tasks (for example construct a plan of the environment).

Cross References

▶ List Scheduling
▶ Load Balancing
▶ Metrical Task Systems
▶ Online Interval Coloring
▶ Online List Update
▶ Packet Switching in Multi-Queue Switches
▶ Packet Switching in Single Buffer
▶ Paging
▶ Robotics
▶ Routing
▶ Work-Function Algorithm for k Servers

Recommended Reading

1. Becchetti, L.: Modeling locality: A probabilistic analysis of LRU and FWF. In: Proceeding 12th European Symposium on Algorithms (ESA) (2004)
2. Borodin, A., Linial, N., Saks, M.E.: An optimal on-line algorithm for metrical task systems. J. ACM **39**, 745–763 (1992)
3. Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. In: Proceeding 35th Annual Symposium on Foundations of Computer Science, pp. 394–400, Santa Fe, NM (1994)
4. Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. SIAM J. Comput. **30**(1), 300–317 (2000)
5. Koutsoupias, E., Papadimitriou, C.H.: On the k-server conjecture. J. ACM **42**(5), 971–983 (1995)
6. Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for on-line problems. In: Proceeding 20th Annual ACM Symposium on the Theory of Computing, pp. 322–333, Chicago, IL (1988)

7. Panagiotou, K., Souza, A.: On adequate performance measures for paging. In: Proceeding 38th annual ACM symposium on Theory of computing, STOC 2006
8. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Comm. ACM. **28**, 202–208 (1985)
9. Young, N.E.: On-Line Paging against Adversarially Biased Random Inputs. J. Algorithms **37**, 218 (2000)

Analyzing Cache Misses

2003; Mehlhorn, Sanders

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

Keywords and Synonyms

Cache analysis

Problem Definition

The problem considered here is *multiple sequence access via cache memory*. Consider the following pattern of memory accesses. k sequences of data, which are stored in disjoint arrays and have a total length of N, are accessed as follows:

```
for $t := 1$ to $N$ do
    select a sequence $s_i \in \{1, \dots k\}$
    work on the current element of sequence $s_i$
    advance sequence $s_i$ to the next element.
```

The aim is to obtain exact (not just asymptotic) closed form upper and lower bounds for this problem. Concurrent accesses to multiple sequences of data are ubiquitous in algorithms. Some examples of algorithms which use this paradigm are distribution sorting, k-way merging, priority queues, permuting and FFT. This entry summarises the analyses of this problem in [3,6].

Caches, Models and Cache Analysis

Modern computers have hierarchical memory which consists of registers, one or more levels of caches, main memory and external memory devices such as disks and tapes. Memory size increases but the speed decreases with distance from the CPU. Hierarchical memory is designed to improve the performance of algorithms by exploiting temporal and spatial locality in data accesses.

Caches are modeled as follows. A cache has m *blocks* each of which holds B data elements. The capacity of the cache is $M = mB$. Data is transferred between one level of cache and the next larger and slower memory in blocks of B elements. A cache is organized as $s = m/a$ *sets* where each set consists of a blocks. Memory at address xB, referred to as memory block x can only be placed in a block in set $x \bmod s$. If $a = 1$ the cache is said to be *direct mapped* and if $a = s$ it is said to be *fully associative*.

If memory block x is accessed and it is not in cache then a *cache miss* occurs and the data in memory block x is brought into cache, incurring a performance penalty. In order to accommodate block x, it is assumed that the least recently used (LRU) or the first used (FIFO) block from the cache set $x \bmod s$ is evicted and this is referred to as the *replacement strategy*. Note that a block may be evicted from a set even though there may be unoccupied blocks in other sets.

Cache analysis is performed for the number of cache misses for a problem with N data elements. To read or write N data elements an algorithm must incur $\Omega(N/B)$ cache misses. These are the *compulsory* or *first reference misses*. In the multiple sequence access via cache memory problem, for given values of M and B, one aim is to find the largest k such that there are $O(N/B)$ cache misses for the N data accesses. It is interesting to analyze cache misses for the important case of direct mapped cache and for the general case of set-associative caches.

A large number of algorithms have been designed on the External Memory Model [9] and these algorithms optimize the number of data transfers between main memory and disk. It seems natural to exploit these algorithms to minimize cache misses, but due to the limited associativity of caches this is not straightforward. In the external memory model data transfers are under programmer control and the multiple sequence access problem has a trivial solution. The algorithm simply chooses $k \le M_e/B_e$, where B_e is the block size and M_e is the capacity of the main memory in the external memory model. For $k \le M_e/B_e$ there are $O(N/B_e)$ accesses to external memory. Since caches are hardware controlled the problem becomes nontrivial. For example, consider the case where the starting addresses of $k > a$ equal length sequences map to the ith element of the same set and the sequences are accessed in a round-robin fashion. On a cache with an LRU or FIFO replacement strategy all sequence accesses will result in a cache miss. Such pathological cases can be overcome by randomizing the starting addresses of the sequences.

Related Problems

A very closely related problem is where accesses to the sequences are interleaved with accesses to a small working array. This occurs in applications such as distribution sorting or matrix multiplication.

Caches can emulate external memory with an optimal replacement policy [1,8] however this requires some constant factor more memory. Since the emulation techniques are software controlled and require modification to the algorithm, rather than selection of parameters, they work well for fairly simple algorithms [4].

Key Results

Theorem 1 [3] *Given an a-way set associative cache with m cache blocks, $s = m/a$ cache sets, cache blocks size B, and LRU or FIFO replacement strategy. Let U_a denote the expected number of cache misses in any schedule of N sequential accesses to k sequences with starting addresses that are at least $(a+1)$-wise independent.*

$$U_1 \le k + \frac{N}{B}\left(1 + (B-1)\frac{k}{m}\right) , \tag{1}$$

$$U_1 \ge \frac{N}{B}\left(1 + (B-1)\frac{k-1}{m+k-1}\right) , \tag{2}$$

$$U_a \le k + \frac{N}{B}\left(1 + (B-1)\left(\frac{k\alpha}{m}\right)^a + \frac{1}{m/(k\alpha)-1} + \frac{k-1}{s-1}\right) \quad \textit{for } k \le \frac{m}{\alpha} , \tag{3}$$

$$U_a \le k + \frac{N}{B}\left(1 + (B-1)\left(\frac{k\beta}{m}\right)^a + \frac{1}{m/(k\beta)-1}\right) \quad \textit{for } k \le \frac{m}{2\beta} , \tag{4}$$

$$U_a \ge \frac{N}{B}\left(1 + (B-1)P_{\text{tail}}\left(k-1, \frac{1}{s}, a\right)\right) - kM , \tag{5}$$

$$U_a \ge \frac{N}{B}\left(1 + (B-1)\left(\frac{(k-a)\alpha}{m}\right)^a\left(1-\frac{1}{s}\right)^k\right) - kM , \tag{6}$$

where $\alpha = \alpha(a) = a/(a!)^{1/a}$, $P_{\text{tail}}(n,p,a) = \sum_{i\ge a}\binom{n}{i} p^i(1-p)^{n-i}$ is the cumulative binomial probability and $\beta := 1 + \alpha(\lceil ax \rceil)$ where $x = x(a) = \inf\{0 < z < 1 : z + z/\alpha(\lceil az \rceil) = 1\}$.

Here $1 \le \alpha < e$ and $\beta(1) = 2$, $\beta(\infty) = 1 + e \approx 3.71$. This analysis assumes that an adversary schedules the accesses to the sequences. For the lower bound the adversary initially advances sequence s_i for $i = 1 \dots k$ by X_i elements, where the X_i are chosen uniformly and independently from $\{0, M-1\}$. The adversary then accesses the sequences in a round-robin manner.

The k in the upper bound accounts for a possible extra block that may be accessed due to randomization of the starting addresses. The $-kM$ term in the lower bound accounts for the fact that cache misses can not be counted when the adversary initially winds forwards the sequences.

The bounds are of the form $pN + c$, where c does not depend on N and p is called the *cache miss probability*. Letting $r = k/m$, the ratio between the number of sequences and the number of cache blocks, the bounds for the cache miss probabilities in Theorem 1 become [3]:

$$p_1 \le (1/B)(1 + (B-1)r) , \tag{7}$$

$$p_1 \ge (1/B)\left(1 + (B-1)\frac{r}{1+r}\right) , \tag{8}$$

$$p_a \le (1/B)(1 + (B-1)(r\alpha)^a + r\alpha + ar) \text{ for } r \le \frac{1}{\alpha} , \tag{9}$$

$$p_a \le (1/B)(1 + (B-1)(r\beta)^a + r\beta) \text{ for } r \le \frac{1}{2\beta} , \tag{10}$$

$$p_a \ge (1/B)\left(1 + (B-1)(r\alpha)^a\left(1-\frac{1}{s}\right)^k\right) . \tag{11}$$

The $1/B$ term accounts for the compulsory or first reference miss, which must be incurred in order to read a block of data from a sequence. The remaining terms account for *conflict misses*, which occur when a block of data is evicted from cache before all its elements have been been scanned. Conflict misses can be reduced by restricting the number of sequences. As r approaches zero the cache miss probabilities approach $1/B$. In general, inequality (4) states that the number of cache misses is $O(N/B)$ if $r \le 1/(2\beta)$ and $(B-1)(r\beta)^a = O(1)$. Both these conditions are satisfied if $k \le m/\max(B^{1/a}, 2\beta)$. So, there are $O(N/B)$ cache misses provided $k = O(m/B^{1/a})$.

The analysis shows that for a direct-mapped cache, where $a = 1$, the upper bound is a factor of $r + 1$ above the lower bound. For $a \ge 2$, the upper bounds and lower bounds are close if $(1 - 1/s)^k \approx$ and $(\alpha + a)r \ll 1$ and both these conditions are satisfied if $k \ll s$.

Rahman and Raman [6] obtain closer upper and lower bounds for average case cache misses assuming the sequences are accessed uniformly randomly on a direct-mapped cache. Sen and Chatterjee [8] also obtain upper and lower bounds assuming the sequences are randomly accessed. Ladner, Fix and LaMarca have analyzed the problem on direct-mapped caches on the independent reference model [2].

Multiple Sequence Access with Additional Working Set

As stated earlier in many applications accesses to sequences are interleaved with accesses to an additional data structure, a *working set*, which determines how a sequence element is to be treated. Assuming that the working set has size at most sB and is stored in contiguous memory locations, the following is an upper bound on the number of cache misses:

Theorem 2 *[3] Let U_a denote the bound on the number of cache misses in Theorem 1 and define $U_0 = N$. With the working set occupying w conflict free memory blocks, the expected number of cache misses arising in the N accesses to the sequence data and any number of accesses to the working set, is bounded by $w + (1 - w/s)U_a + 2(w/s)U_{a-1}$.*

On a direct mapped cache, for $i = 1, \ldots, k$, if sequence i is accessed with probability p_i independently of all previous accesses and is followed by an access to element i of the working set then the following are upper and lower bounds for the number of cache misses:

Theorem 3 *[6] In a direct-mapped cache with m cache blocks each of B elements, if sequence i, for $i = 1, \ldots, k$, is accessed with probability p_i and block j of the working set, for $j = 1, \ldots, k/B$, is accessed with probability P_j then the expected number of cache misses in N sequence accesses is at most $N(p_s + p_w) + k(1 + 1/B)$, where:*

$$p_s \le \frac{1}{B} + \frac{k}{mB} + \frac{B-1}{mB} \sum_{i=1}^{k} \left(\sum_{j=1}^{k/B} \frac{p_i P_j}{p_i + P_j} + \frac{B-1}{B} \sum_{j=1}^{k} \frac{p_i p_j}{p_i + p_j} \right) ,$$

$$p_w \le \frac{k}{B^2 m} + \frac{B-1}{mB} \sum_{i=1}^{k/B} \sum_{j=1}^{k} \frac{P_i p_j}{P_i + p_j} .$$

Theorem 4 *[6] In a direct-mapped cache with m cache blocks each of B elements, if sequence i, for $i = 1, \ldots, k$, is accessed with probability $p_i \ge 1/m$ then the expected number of cache misses in N sequence accesses is at least $Np_s + k$, where:*

$$\begin{aligned} p_s \ge{} & \frac{1}{B} + \frac{k(2m-k)}{2m^2} + \frac{k(k-3m)}{2Bm^2} - \frac{1}{2Bm} - \frac{k}{2B^2 m} \\ & + \frac{B(k-m) + 2m - 3k}{Bm^2} \sum_{i=1}^{k} \sum_{j=1}^{k} \frac{(p_i)^2}{p_i + p_j} \\ & + \frac{(B-1)^2}{B^3 m^2} \sum_{i=1}^{k} p_i \left[\sum_{j=1}^{k} \frac{p_i(1 - p_i - p_j)}{(p_i + p_j)^2} - \frac{B-1}{2} \right. \\ & \left. \sum_{j=1}^{k} \sum_{l=1}^{k} \frac{p_i}{p_i + p_j + p_l - p_j p_l} \right] - O\left(e^{-B}\right) . \end{aligned}$$

The lower bound ignores the interaction with the working set, since this can only increase the number of cache misses.

In Theorem 3 and Theorem 4 p_s is the probability of a cache miss for a sequence access and in Theorem 3 p_w is the probability of a cache miss for an accesses to the working set.

If the sequences are accessed uniformly randomly, then using Theorem 3 and Theorem 4, the ratio between the upper and lower bound is $3/(3 - r)$, where $r = k/m$. So for uniformly random data the lower bound is within a factor of about 3/2 of the upper bound when $k \le m$ and is much closer when $k \ll m$.

Applications

Numerous algorithms have been developed on the external memory model which access multiple sequences of data, such as merge-sort, distribution sort, priority queues, radix sorting. These analyzes are important as they allow initial parameter choices to be made for cache memory algorithms.

Open Problems

The analyzes assume that the starting addresses of the sequences are randomized and current approaches to allocating random starting addresses waste a lot of virtual address space [3]. An open problem is to find a good online scheme to randomize the starting addresses of arbitrary length sequences.

Experimental Results

The cache model is a powerful abstraction of real caches, however modern computer architectures have complex internal memory hierarchies, with registers, multiple levels of caches and *translation-lookaside-buffers* (TLB). Cache miss penalties are not of the same magnitude as the cost of disk accesses, so an algorithm may perform better by allowing conflict misses to increase in order to reduce computation costs and compulsory misses, by reducing the number of passes over the data. This means that in practice cache analyzes is used to choose an initial value of k which is then fine tuned for the platform and algorithm [4,5,7,10].

For distribution sorting, in [4] a heuristic was considered for selecting k and equations for approximate cache misses were obtained. These equations were shown to be very accurate in practice.

Cross References

- ► Cache-Oblivious Model
- ► Cache-Oblivious Sorting
- ► External Sorting and Permuting
- ► I/O-model

Recommended Reading

1. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: Proc. of 40th Annual Symposium on Foundations of Computer Science (FOCS'99), pp. 285–298 IEEE Computer Society, Washington D.C. (1999)
2. Ladner, R.E., Fix, J.D., LaMarca, A.: Cache performance analysis of traversals and random accesses. In: Proc. of 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1999), pp. 613–622 Society for Industrial and Applied Mathematics, Philadelphia (1999)
3. Mehlhorn, K., Sanders, P.: Scanning multiple sequences via cache memory. Algorithmica **35**, 75–93 (2003)
4. Rahman, N., Raman, R.: Adapting radix sort to the memory hierarchy. ACM J. Exp. Algorithmics **6**, Article 7 (2001)
5. Rahman, N., Raman, R.: Analysing cache effects in distribution sorting. ACM J. Exp. Algorithmics **5**, Article 14 (2000)
6. Rahman, N., Raman, R.: Cache analysis of non-uniform distribution sorting algorithms. (2007) http://www.citebase.org/abstract?id=oai:arXiv.org:0706.2839 Accessed 13 August 2007 Preliminary version in: Proc. of 8th Annual European Symposium on Algorithms (ESA 2000). LNCS, vol. 1879, pp. 380–391. Springer, Berlin Heidelberg (2000)
7. Sanders, P.: Fast priority queues for cached memory. ACM J. Exp. Algorithmics **5**, Article 7 (2000)
8. Sen, S., Chatterjee, S.: Towards a theory of cache-efficient algorithms. In: Proc. of 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000), pp. 829–838. Society for Industrial and Applied Mathematics (2000)
9. Vitter, J.S.: External memory algorithms and data structures: dealing with massive data. ACM Comput. Surv. **33**, 209–271 (2001)
10. Wickremesinghe, R., Arge, L., Chase, J.S., Vitter, J.S.: Efficient sorting using registers and caches. ACM J. Exp. Algorithmics **7**, 9 (2002)

Applications of Geometric Spanner Networks

2002; Gudmundsson, Levcopoulos, Narasimhan, Smid

JOACHIM GUDMUNDSSON[1], GIRI NARASIMHAN[2], MICHIEL SMID[3]

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

[2] School of Computing and Information Science, Florida International University, Miami, FL, USA

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

Keywords and Synonyms

Stretch factor

Problem Definition

Given a geometric graph in d-dimensional space, it is useful to preprocess it so that distance queries, exact or approximate, can be answered efficiently. Algorithms that can report distance queries in constant time are also referred to as "distance oracles". With unlimited preprocessing time and space, it is clear that exact distance oracles can be easily designed. This entry sheds light on the design of approximate distance oracles with limited preprocessing time and space for the family of geometric graphs with constant dilation.

Notation and Definitions

If p and q are points in $\mathbb{R}^d$, then the notation $|pq|$ is used to denote the Euclidean distance between p and q; the notation $\delta_G(p, q)$ is used to denote the Euclidean length of a shortest path between p and q in a geometric network G. Given a constant $t > 1$, a graph G with vertex set S is a t-spanner for S if $\delta_G(p, q) \le t|pq|$ for any two points p and q of S. A t-spanner network is said to have *dilation* (or *stretch factor*) t. A $(1 + \varepsilon)$-approximate shortest path between p and q is defined to be any path in G between p and q having length Δ, where $\delta_G(p, q) \le \Delta \le (1 + \varepsilon)\delta_G(p, q)$. For a comprehensive overview of geometric spanners, see the book by Narasimhan and Smid [13].

All networks considered in this entry are simple and undirected. The model of computation used is the traditional algebraic computation tree model with the added power of indirect addressing. In particular, the algorithms presented here do not use the non-algebraic floor function as a unit-time operation. The problem is formalized below.

Problem 1 (Distance Oracle) *Given an arbitrary real constant $\varepsilon > 0$, and a geometric graph G in d-dimensional Euclidean space with constant dilation t, design a data structure that answers $(1 + \varepsilon)$-approximate shortest path length queries in constant time.*

The data structure can also be applied to solve several other problems. These include (a) the problem of reporting approximate distance queries between vertices in a planar polygonal domain with "rounded" obstacles, (b) query versions of *closest pair* problems, and (c) the efficient computation of the approximate dilations of geometric graphs.

Survey of Related Research

The design of efficient data structures for answering distance queries for general (non-geometric) networks was considered by Thorup and Zwick [15] (unweighted general graphs), Baswanna and Sen [3] (weighted general graphs, i. e., arbitrary metrics), and Arikati et al. [2] and Thorup [14] (weighted planar graphs).

For the geometric case, variants of the problem have been considered in a number of papers (for a recent paper see, for example, Chen et al. [5]). Work on the approximate version of these variants can also be found in many articles (for a recent paper see, for example, Agarwal et al. [1]). The focus of this entry are the results reported in the work of Gudmundsson et al. [9,10,11,12].

Key Results

The main result of this entry is the existence of approximate distance oracle data structures for geometric networks with constant dilation (see "Theorem 4" below). As preprocessing, the network is "pruned" so that it only has a linear number of edges. The data structure consists of a series of "cluster graphs" of increasing coarseness each of which helps answer approximate queries for pairs of points with interpoint distances of different scales. In order to pinpoint the appropriate cluster graph to search in for a given query, the data structure uses the bucketing tool described below. The idea of using cluster graphs to speed up geometric algorithms was first introduced by Das and Narasimhan [6] and later used by Gudmundsson et al. [8] to design an efficient algorithm to compute $(1 + \varepsilon)$-spanners. Similar ideas were explored by Gao et al. [7] for applications to the design of mobile networks.

Pruning

If the input geometric network has a superlinear number of edges, then the preprocessing step for the distance oracle data structure involves efficiently "pruning" the network so that it has only a linear number of edges. The pruning may result in a small increase of the dilation of the spanner. The following theorem was proved by Gudmundsson et al. [12].

Theorem 1 *Let $t > 1$ and $\varepsilon' > 0$ be real constants. Let S be a set of n points in $\mathbb{R}^d$, and let $G = (S, E)$ be a t-spanner for S with m edges. There exists an algorithm to compute in $O(m + n \log n)$ time, a $(1 + \varepsilon')$-spanner of G having $O(n)$ edges and whose weight is $O(wt(MST(S)))$.*

The pruning step requires the following technical theorem proved by Gudmundsson et al. [12].

Theorem 2 *Let S be a set of n points in $\mathbb{R}^d$, and let $c \geq 7$ be an integer constant. In $O(n \log n)$ time, it is possible to compute a data structure D(S) consisting of:*

1. *a sequence $L_1, L_2, \dots, L_\ell$ of real numbers, where $\ell = O(n)$, and*
2. *a sequence $S_1, S_2, \dots, S_\ell$ of subsets of S satisfying $\sum_{i=1}^{\ell} |S_i| = O(n)$,*

such that the following holds. For any two distinct points p and q of S, it is possible to compute in O(1) time an index i with $1 \leq i \leq \ell$ and two points x and y in S_i such that (a) $L_i/n^{c+1} \leq |xy| < L_i$, and (b) both $|px|$ and $|qy|$ are less than $|xy|/n^{c-2}$.

Despite its technical nature, the above theorem is of fundamental importance to this work. In particular, it helps to deal with networks where the interpoint distances are not confined to a polynomial range, i. e., there are pairs of points that are very close to each other and very far from each other.

Bucketing

Since the model of computation assumed here does not allow the use of floor functions, an important component of the algorithm is a "bucketing tool" that allows (after appropriate preprocessing) constant-time computation of a quantity referred to as BINDEX, which is defined to be the floor of the logarithm of the interpoint distance between any pair of input points.

Theorem 3 *Let S be a set of n points in $\mathbb{R}^d$ that are contained in the hypercube $(0, n^k)^d$, for some positive integer constant k, and let ε be a positive real constant. The set S can be preprocessed in $O(n \log n)$ time into a data structure of size $O(n)$, such that for any two points p and q of S, with $|pq| \geq 1$, it is possible to compute in constant time the quantity $BIndex_\varepsilon(p, q) = \lfloor \log_{1+\varepsilon} |pq| \rfloor$.*

The constant-time computation mentioned in Theorem 3 is achieved by reducing the problem to one of answering least common ancestor queries for pairs of nodes in a tree, a problem for which constant-time solutions were devised most recently by Bender and Farach-Colton [4].

Main Results

Using the bucketing and the pruning tools, and using the algorithms described by Gudmundsson et al. [11], the following theorem can be proved.

Theorem 4 *Let $t > 1$ and $\varepsilon > 0$ be real constants. Let S be a set of n points in $\mathbb{R}^d$, and let $G = (S, E)$ be a t-spanner for S with m edges. The graph G can be preprocessed into a data structure of size $O(n \log n)$ in time $O(m + n \log n)$, such that for any pair of query points $p, q \in S$, it is possible to compute a $(1+\varepsilon)$-approximation of the shortest-path distance in G between p and q in $O(1)$ time. Note that all the big-Oh notations hide constants that depend on d, t and ε.*

Additionally, if the traditional algebraic model of computation (without indirect addressing) is assumed, the following weaker result can be proved.

Theorem 5 *Let S be a set of n points in $\mathbb{R}^d$, and let $G = (S, E)$ be a t-spanner for S, for some real constant $t > 1$, having m edges. Assuming the algebraic model of computation, in $O(m \log\log n + n \log^2 n)$ time, it is possible to preprocess G into a data structure of size $O(n \log n)$, such that for any two points p and q in S, a $(1 + \varepsilon)$-approximation of the shortest-path distance in G between p and q can be computed in $O(\log\log n)$ time.*

Applications

As mentioned earlier, the data structure described above can be applied to several other problems. The first application deals with reporting distance queries for a planar domain with polygonal obstacles. The domain is further constrained to be t-rounded, which means that the length of the shortest obstacle-avoiding path between any two points in the input point set is at most t times the Euclidean distance between them. In other words, the visibility graph is required to be a t-spanner for the input point set.

Theorem 6 *Let $\mathcal{F}$ be a t-rounded collection of polygonal obstacles in the plane of total complexity n, where t is a positive constant. One can preprocess $\mathcal{F}$ in $O(n \log n)$ time into a data structure of size $O(n \log n)$ that can answer obstacle-avoiding $(1+\varepsilon)$-approximate shortest path length queries in time $O(\log n)$. If the query points are vertices of $\mathcal{F}$, then the queries can be answered in O(1) time.*

The next application of the distance oracle data structure includes query versions of *closest pair* problems, where the queries are confined to specified subset(s) of the input set.

Theorem 7 *Let $G = (S, E)$ be a geometric graph on n points and m edges, such that G is a t-spanner for S, for some constant $t > 1$. One can preprocess G in time $O(m+n \log n)$ into a data structure of size $O(n \log n)$ such that given a query subset S' of S, a $(1 + \varepsilon)$-approximate closest pair in S' (where distances are measured in G) can be computed in time $O(|S'| \log |S'|)$.*

Theorem 8 *Let $G = (S, E)$ be a geometric graph on n points and m edges, such that G is a t-spanner for S, for some constant $t > 1$. One can preprocess G in time $O(m+n \log n)$ into a data structure of size $O(n \log n)$ such that given two disjoint query subsets X and Y of S, a $(1 + \varepsilon)$-approximate bichromatic closest pair (where distances are measured in G) can be computed in time $O((|X| + |Y|) \log(|X| + |Y|))$.*

The last application of the distance oracle data structure includes the efficient computation of the approximate dilations of geometric graphs.

Theorem 9 *Given a geometric graph on n vertices with m edges, and given a constant C that is an upper bound on the dilation t of G, it is possible to compute a $(1 + \varepsilon)$-approximation to t in time $O(m + n \log n)$.*

Open Problems

Two open problems remain unanswered.

1. Improve the space utilization of the distance oracle data structure from $O(n \log n)$ to $O(n)$.
2. Extend the approximate distance oracle data structure to report not only the approximate distance, but also the approximate shortest path between the given query points.

Cross References

- ▶ All Pairs Shortest Paths in Sparse Graphs
- ▶ All Pairs Shortest Paths via Matrix Multiplication
- ▶ Geometric Spanners
- ▶ Planar Geometric Spanners
- ▶ Sparse Graph Spanners
- ▶ Synchronizers, Spanners

Recommended Reading

1. Agarwal, P.K., Har-Peled, S., Karia, M.: Computing approximate shortest paths on convex polytopes. In: Proceedings of the 16th ACM Symposium on Computational Geometry, pp. 270–279. ACM Press, New York (2000)
2. Arikati, S., Chen, D.Z., Chew, L.P., Das, G., Smid, M., Zaroliagis, C.D.: Planar spanners and approximate shortest path queries among obstacles in the plane. In: Proceedings of the 4th Annual European Symposium on Algorithms. Lecture Notes in

Computer Science, vol. 1136, Berlin, pp. 514–528. Springer, London (1996)

3. Baswana, S., Sen, S.: Approximate distance oracles for unweighted graphs in $\tilde{O}(n^2)$ time. In: Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms, pp. 271–280. ACM Press, New York (2004)
4. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Proceedings of the 4th Latin American Symposium on Theoretical Informatics. Lecture Notes in Computer Science, vol. 1776, Berlin, pp. 88–94. Springer, London (2000)
5. Chen, D.Z., Daescu, O., Klenk, K.S.: On geometric path query problems. Int. J. Comput. Geom. Appl. **11**, 617–645 (2001)
6. Das, G., Narasimhan, G.: A fast algorithm for constructing sparse Euclidean spanners. Int. J. Comput. Geom. Appl. **7**, 297–315 (1997)
7. Gao, J., Guibas, L.J., Hershberger, J., Zhang, L., Zhu, A.: Discrete mobile centers. Discrete Comput. Geom. **30**, 45–63 (2003)
8. Gudmundsson, J., Levcopoulos, C., Narasimhan, G.: Fast greedy algorithms for constructing sparse geometric spanners. SIAM J. Comput. **31**, 1479–1500 (2002)
9. Gudmundsson, J., Levcopoulos, C., Narasimhan, G., Smid, M.: Approximate distance oracles for geometric graphs. In: Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms, pp. 828–837. ACM Press, New York (2002)
10. Gudmundsson, J., Levcopoulos, C., Narasimhan, G., Smid, M.: Approximate distance oracles revisited. In: Proceedings of the 13th International Symposium on Algorithms and Computation. Lecture Notes in Computer Science, vol. 2518, Berlin, pp. 357–368. Springer, London (2002)
11. Gudmundsson, J., Levcopoulos, C., Narasimhan, G., Smid, M.: Approximate distance oracles for geometric spanners, ACM Trans. Algorithms (2008). To Appear
12. Gudmundsson, J., Narasimhan, G., Smid, M.: Fast pruning of geometric spanners. In: Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science, vol. 3404, Berlin, pp. 508–520. Springer, London (2005)
13. Narasimhan, G., Smid, M.: Geometric Spanner Networks, Cambridge University Press, Cambridge, UK (2007)
14. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. J. ACM **51**, 993–1024 (2004)
15. Thorup, M., Zwick, U.: Approximate distance oracles. In: Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing, pp. 183–192. ACM Press, New York (2001)

Approximate Dictionaries

2002; Buhrman, Miltersen, Radhakrishnan, Venkatesh

VENKATESH SRINIVASAN
Department of Computer Science, University of Victoria,
Victoria, BC, Canada

Keywords and Synonyms

Static membership; Approximate membership

Problem Definition

The Problem and the Model

A static data structure problem consists of a set of data D, a set of queries Q, a set of answers A, and a function $f: D \times Q \to A$. The goal is to store the data succinctly so that any query can be answered with only a few probes to the data structure. *Static membership* is a well-studied problem in data structure design [1,4,7,8,12,13,16].

Definition 1 (Static Membership) In the static membership problem, one is given a subset S of at most n keys from a universe $U = \{1, 2, \ldots, m\}$. The task is to store S so that queries of the form "Is u in S?" can be answered by making few accesses to the memory.

A natural and general model for studying any data structure problem is the *cell probe model* proposed by Yao [16].

Definition 2 (Cell Probe Model) An (s, w, t) cell probe scheme for a static data structure problem $f: D \times Q \to A$ has two components: a storage scheme and a query scheme. The storage scheme stores the data $d \in D$ as a table $T[d]$ of s cells, each cell of word size w bits. The storage scheme is deterministic. Given a query $q \in Q$, the query scheme computes $f(d, q)$ by making at most t probes to $T[d]$, where each probe reads one cell at a time, and the probes can be adaptive. In a deterministic cell probe scheme, the query scheme is deterministic. In a randomized cell probe scheme, the query scheme is randomized and is allowed to err with a small probability.

Buhrman et al. [2] study the complexity of the static membership problem in the *bitprobe* model. The bitprobe model is a variant of the cell probe model in which each cell holds just a single bit. In other words, the word size w is 1. Thus, in this model, the query algorithm is given bitwise access to the data structure. The study of the membership problem in the bitprobe model was initiated by Minsky and Papert in their book *Perceptrons* [12]. However, they were interested in *average*-case upper bounds for this problem, while this work studies *worst*-case bounds for the membership problem.

Observe that if a scheme is required to store sets of size at most n, then it must use at least $\lceil \log \sum_{i \le n} \binom{m}{i} \rceil$ number of bits. If $n \le m^{1-\Omega(1)}$, this implies that the scheme must store $\Omega(n \log m)$ bits (and therefore use $\Omega(n)$ cells). The goal in [2] is to obtain a scheme that answers queries uses only a constant number of bitprobes and at the same time uses a table of $O(n \log m)$ bits.

Related Work

The static membership problem has been well studied in the cell probe model, where each cell is capable of holding one element of the universe. That is, $w = O(\log m)$. In a seminal paper, Fredman et al. [8] proposed a scheme for the static membership problem in the cell probe model with word size $O(\log m)$ that used a constant number of probes and a table of size $O(n)$. This scheme will be referred to as the FKS scheme. Thus, up to constant factors, the FKS scheme uses optimal space and number of cell probes. In fact, Fiat et al. [7], Brodnik and Munro [1], and Pagh [13] obtain schemes that use space (in bits) that is within a small additive term of $\lceil \log \sum_{i \le n} \binom{m}{i} \rceil$ and yet answer queries by reading at most a constant number of cells. Despite all these fundamental results for the membership problem in the cell probe model, very little was known about the bitprobe complexity of static membership prior to the work in [2].

Key Results

Buhrman et al. investigate the complexity of the static membership problem in the bitprobe model. They study

- Two-sided error randomized schemes that are allowed to err on positive instances as well as negative instances (that is, these schemes can say 'No' with a small probability when the query element u is in the set S and 'Yes' when it is not);
- One-sided error randomized schemes where the errors are restricted to negative instances alone (that is, these schemes never say 'No' when the query element u is in the set S);
- Deterministic schemes in which no errors are allowed.

The main techniques used in [2] are based on two-colorings of special set systems that are related to the r-cover-free family of sets considered in [3,5,9]. The reader is referred to [2] for further details.

Randomized Schemes with Two-Sided Error

The main result in [2] shows that there are randomized schemes that use *just one bitprobe* and yet use space close to the information theoretic lower bound of $\Omega(n \log m)$ bits.

Theorem 1 *For any $0 < \epsilon \le \frac{1}{4}$, there is a scheme for storing subsets S of size at most n of a universe of size m using $O(\frac{n}{\epsilon^2} \log m)$ bits so that any membership query "Is $u \in S$?" can be answered with an error probability of at most ϵ by a randomized algorithm that probes the memory at just one location determined by its coin tosses and the query element u.*

Note that randomization is allowed only in the query algorithm. It is still the case that for each set S, there is exactly one associated data structure $T(S)$. It can be shown that deterministic schemes that answer queries using a single bitprobe need m bits of storage (see the remarks following Theorem 4). Theorem 1 shows that, by allowing randomization, this bound (for constant ϵ) can be reduced to $O(n \log m)$ bits. This space is within a constant factor of the information theoretic bound for n sufficiently small. Yet the randomized scheme answers queries using a single bitprobe.

Unfortunately, the construction above does not permit us to have subconstant error probability and still use optimal space. Is it possible to improve the result of Theorem 1 further and design such a scheme? [2] shows that this is not possible: if ϵ is made subconstant, then the scheme must use more than $n \log m$ space.

Theorem 2 *Suppose $\frac{n}{m^{1/3}} \le \epsilon \le \frac{1}{4}$. Then, any two-sided ϵ-error randomized scheme that answers queries using one bitprobe must use space $\Omega(\frac{n}{\epsilon \log(1/\epsilon)} \log m)$.*

Randomized Schemes with One-Sided Error

Is it possible to have any savings in space if the query scheme is expected to make only one-sided errors? The following result shows it is possible if the error is allowed only on negative instances.

Theorem 3 *For any $0 < \epsilon \le \frac{1}{4}$, there is a scheme for storing subsets S of size at most n of a universe of size m using $O((\frac{n}{\epsilon})^2 \log m)$ bits so that any membership query "Is $u \in S$?" can be answered with error probability at most ϵ by a randomized algorithm that makes a single bitprobe to the data structure. Furthermore, if $u \in S$, the probability of error is 0.*

Though this scheme does not operate with optimal space, it still uses significantly less space than a bitvector. However, the dependence on n is quadratic, unlike in the two-sided scheme where it was linear. [2] shows that this scheme is essentially optimal: there is necessarily a quadratic dependence on $\frac{n}{\epsilon}$ for any scheme with one-sided error.

Theorem 4 *Suppose $\frac{n}{m^{1/3}} \le \epsilon \le \frac{1}{4}$. Consider the static membership problem for sets S of size at most n from a universe of size m. Then, any scheme with one-sided error ϵ that answers queries using at most one bitprobe must use $\Omega(\frac{n^2}{\epsilon^2 \log(n/\epsilon)} \log m)$ bits of storage.*

Remark One could also consider one-probe, one-sided error schemes that only make errors on positive instances. That is, no error is made for query elements *not* in the set S.

In this case, [2] shows that randomness does not help at all: such a scheme must use m bits of storage.

The following result shows that the space requirement can be reduced further in one-sided error schemes if more probes are allowed.

Theorem 5 *Suppose $0 < \delta < 1$. There is a randomized scheme with one-sided error $n^{-\delta}$ that solves the static membership problem using $O(n^{1+\delta} \log m)$ bits of storage and $O(\frac{1}{\delta})$ bitprobes.*

Deterministic Schemes

In contrast to randomized schemes, Buhrman et al. show that deterministic schemes exhibit a time-space tradeoff behavior.

Theorem 6 *Suppose a deterministic scheme stores subsets of size n from a universe of size m using s bits of storage and answers membership queries with t bitprobes to memory. Then, $\binom{m}{n} \leq \max_{i \leq nt} \binom{2s}{i}$.*

This tradeoff result has an interesting consequence. Recall that the FKS hashing scheme is a data structure for storing sets of size at most n from a universe of size m using $O(n \log m)$ bits, so that membership queries can be answered using $O(\log m)$ bitprobes. As a corollary of the tradeoff result, [2] shows that the FKS scheme makes an optimal number of bitprobes, within a constant factor, for this amount of space.

Corollary 1 *Let $\epsilon > 0, c \geq 1$ be any constants. There is a constant $\delta > 0$ so that the following holds. Let $n \leq m^{1-\epsilon}$ and let a scheme for storing sets of size at most n of a universe of size m as data structures of at most $cn \log m$ bits be given. Then, any deterministic algorithm answering membership queries using this structure must make at least $\delta \log m$ bitprobes in the worst case.*

From Theorem 6 it also follows that any deterministic scheme that answers queries using t bitprobes must use space at least $ntm^{\Omega(1/t)}$ in the worst case. The final result shows the existence of schemes which almost match the lower bound.

Theorem 7

1. *There is a nonadaptive scheme that stores sets of size at most n from a universe of size m using $O(ntm^{\frac{2}{t+1}})$ bits and answers queries using $2t + 1$ bitprobes. This scheme is nonexplicit.*
2. *There is an explicit adaptive scheme that stores sets of size at most n from a universe of size m using $O(m^{1/t} n \log m)$ bits and answers queries using $O(\log n + \log \log m) + t$ bitprobes.*

Applications

The results in [2] have interesting connections to questions in coding theory and communication complexity. In the framework of coding theory, the results in [2] can be viewed as constructing locally decodable source codes, analogous to the locally decodable channel codes of [10]. Theorems 1–4 can also be viewed as giving tight bounds for the following communication complexity problem (as pointed out in [11]): Alice gets $u \in \{1, \ldots, m\}$, Bob gets $S \subseteq \{1, \ldots, m\}$ of size at most n, and Alice sends a single message to Bob after which Bob announces whether $u \in S$. See [2] for further details.

Recommended Reading

1. Brodnik, A., Munro, J.I.: Membership in constant time and minimum space. In: Lecture Notes in Computer Science, vol. 855, pp. 72–81, Springer, Berlin (1994). Final version: Membership in Constant Time and Almost-Minimum Space. SIAM J. Comput. **28**(5), 1627–1640 (1999)
2. Buhrman, H., Miltersen, P.B., Radhakrishnan, J., Venkatesh, S.: Are bitvectors optimal? SIAM J. Comput. **31**(6), 1723–1744 (2002)
3. Dyachkov, A.G., Rykov, V.V.: Bounds on the length of disjunctive codes. Problemy Peredachi Informatsii **18**(3), 7–13 (1982)
4. Elias, P., Flower, R.A.: The complexity of some simple retrieval problems. J. Assoc. Comput. Mach. **22**, 367–379 (1975)
5. Erdös, P., Frankl, P., Füredi, Z.: Families of finite sets in which no set is covered by the union of r others. Isr. J. Math. **51**, 79–89 (1985)
6. Fiat, A., Naor, M.: Implicit $O(1)$ probe search. SIAM J. Comput. **22**, 1–10 (1993)
7. Fiat, A., Naor, M., Schmidt, J.P., Siegel, A.: Non-oblivious hashing. J. Assoc. Comput. Mach. **31**, 764–782 (1992)
8. Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a sparse table with $O(1)$ worst case access time. J. Assoc. Comput. Mach. **31**(3), 538–544 (1984)
9. Füredi, Z.: On r-cover-free families. J. Comb. Theory, Series A **73**, 172–173 (1996)
10. Katz, J., Trevisan, L.: On the efficiency of local decoding procedures for error-correcting codes. In: Proceedings of STOC'00, pp. 80–86
11. Miltersen, P.B., Nisan, N., Safra, S., Wigderson, A.: On data structures and asymmetric communication complexity. J. Comput. Syst. Sci. **57**, 37–49 (1998)
12. Minsky, M., Papert, S.: Perceptrons. MIT Press, Cambridge (1969)
13. Pagh, R.: Low redundancy in static dictionaries with $O(1)$ lookup time. In: Proceedings of ICALP '99. LNCS, vol. 1644, pp. 595–604. Springer, Berlin (1999)
14. Ruszinkó, M. On the upper bound of the size of r-cover-free families. J. Comb. Theory, Ser. A **66**, 302–310 (1984)
15. Ta-Shma, A.: Explicit one-probe storing schemes using universal extractors. Inf. Proc. Lett. **83**(5), 267–274 (2002)
16. Yao, A.C.C.: Should tables be sorted? J. Assoc. Comput. Mach. **28**(3), 615–628 (1981)

Approximate Dictionary Matching

▶ Dictionary Matching and Indexing (Exact and with Errors)

Approximate Maximum Flow Construction

▶ Randomized Parallel Approximations to Max Flow

Approximate Membership

▶ Approximate Dictionaries

Approximate Nash Equilibrium

▶ Non-approximability of Bimatrix Nash Equilibria

Approximate Periodicities

▶ Approximate Tandem Repeats

Approximate Regular Expression Matching

1995; Wu, Manber, Myers

GONZALO NAVARRO
Department of Computer Science, University of Chile,
Santiago, Chile

Keywords and Synonyms

Regular expression matching allowing errors or differences

Problem Definition

Given a *text string* $T = t_1 t_2 \dots t_n$ and a *regular expression* R of length m denoting language $\mathcal{L}(R)$, over an alphabet Σ of size σ, and given a *distance function* among strings d and a *threshold* k, the *approximate regular expression matching (AREM)* problem is to find all the text positions that finish a so-called *approximate occurrence* of R in T, that is, compute the set $\{j, \exists i, 1 \leq i \leq j, \exists P \in \mathcal{L}(R), d(P, t_i \dots t_j) \leq k\}$. T, R, and k are given together, whereas the algorithm can be tailored for a specific d.

This entry focuses on the so-called *weighted edit distance*, which is the minimum sum of weights of a sequence of operations converting one string into the other. The operations are insertions, deletions, and substitutions of characters. The weights are positive real values associated to each operation and characters involved. The weight of deleting a character c is written $w(c \to \epsilon)$, that of inserting c is written $w(\epsilon \to c)$, and that of substituting c by $c' \neq c$ is written $w(c \to c')$. It is assumed $w(c \to c) = 0$ for all $c \in \Sigma \cup \epsilon$ and the triangle inequality, that is, $w(x \to y) + w(y \to z) \geq w(x \to z)$ for any $x, y, z \in \Sigma \cup \{\epsilon\}$. As the distance may be asymmetric, it is also fixed that that $d(A, B)$ is the cost of converting A into B. For simplicity and practicality $m = o(n)$ is assumed in this entry.

Key Results

The most versatile solution to the problem [3] is based on a graph model of the distance computation process. Assume the regular expression R is converted into a nondeterministic finite automaton (NFA) with $O(m)$ states and transitions using Thompson's method [8]. Take this automaton as a directed graph $G(V, E)$ where edges are labeled by elements in $\Sigma \cup \{\epsilon\}$. A directed and weighted graph $\mathcal{G}$ is built to solve the AREM problem. $\mathcal{G}$ is formed by putting $n + 1$ copies of G, $G_0, G_1, \dots, G_n$, and connecting them with weights so that the distance computation reduces to finding shortest paths in $\mathcal{G}$.

More formally, the nodes of $\mathcal{G}$ are $\{v_i, v \in V, 0 \leq i \leq n\}$, so that v_i is the copy of node $v \in V$ in graph G_i. For each edge $u \xrightarrow{c} v$ in E, $c \in \Sigma \cup \{\epsilon\}$, the following edges are added to graph $\mathcal{G}$:

$$\begin{array}{lll}
u_i \to v_i\,, & \text{with weight } w(c \to \epsilon)\,, & 0 \leq i \leq n\,,\\
u_i \to u_{i+1}\,, & \text{with weight } w(\epsilon \to t_{i+1})\,, & 0 \leq i < n\,,\\
u_i \to v_{i+1}\,, & \text{with weight } w(c \to t_{i+1})\,, & 0 \leq i < n\,.
\end{array}$$

Assume for simplicity that G has initial state s and a unique final state f (this can always be arranged). As defined, the shortest path in $\mathcal{G}$ from s_0 to f_n gives the smallest distance between T and a string in $\mathcal{L}(R)$. In order to adapt the graph to the AREM problem, the weights of the edges between s_i and s_{i+1} are modified to be zero.

Then, the AREM problem is reduced to computing shortest paths. It is not hard to see that $\mathcal{G}$ can be topologically sorted so that all the paths to nodes in G_i are computed before all those to G_{i+1}. This way, it is not hard to solve this shortest path problem in $O(mn \log m)$ time and $O(m)$ space. Actually, if one restricts the problem to the particular case of *network expressions*, which are regular expressions without Kleene closure, then G has no loops and the shortest path computation can be done in $O(mn)$ time, and even better on average [2].

The most delicate part in achieving $O(mn)$ time for general regular expressions [3] is to prove that, given the types of loops that arise in the NFAs of regular expressions, it is possible to compute the distances correctly within each G_i by (a) computing them in a topological order of G_i without considering the *back edges* introduced by Kleene closures; (b) updating path costs by using the back edges once; (c) updating path costs once more in topological order ignoring back edges again.

Theorem 1 (Myers and Miller 1989 [3]) *There exists an $O(mn)$ worst-case time solution to the AREM problem under weighted edit distance.*

It is possible to do better when the weights are integer-valued, by exploiting the unit-cost RAM model through a four-Russian technique [10]. The idea is as follows. Take a small subexpression of R, which produces an NFA that will translate into a small subgraph of each G_i. At the time of propagating path costs within this automaton, there will be a counter associated to each node (telling the current shortest path from s_0). This counter can be reduced to a number in $[0, k+1]$, where $k+1$ means "more than k". If the small NFA has r states, $r\lceil \log_2(k+2)\rceil$ bits are needed to fully describe the counters of the corresponding subgraph of G_i. Moreover, given an initial set of values for the counters, it is possible to precompute all the propagation that will occur within the same subgraph of G_i, in a table having $2^{r\lceil \log_2(k+2)\rceil}$ entries, one per possible configuration of counters. It is sufficient that $r < \alpha \log_{k+2} n$ for some $\alpha < 1$ to make the construction and storage cost of those tables $o(n)$. With the help of those tables, all the propagation within the subgraph can be carried out in constant time. Similarly, the propagation of costs to the same subgraph at G_{i+1} can also be precomputed in tables, as it depends only on the current counters in G_i and on text character t_{i+1}, for which there are only σ alternatives.

Now, take all the subtrees of R of maximum size not exceeding r and preprocess them with the technique above. Convert each such subtree into a leaf in R labeled by a special character a_A, associated to the corresponding small NFA A. Unless there are consecutive Kleene closures in R, which can be simplified as $R^{**} = R^*$, the size of R after this transformation is $O(m/r)$. Call R' the transformed regular expression. One essentially applies the technique of Theorem 1 to R', taking care of how to deal with the special leaves that correspond to small NFAs. Those leaves are converted by Thompson's construction into two nodes linked by an edge labeled a_A. When the path cost propagation process reaches the source node of an edge labeled a_A with cost c, one must update the counter of the initial state of NFA A to c (or $k+1$ if $c > k$). One then uses the four-Russians table to do all the cost propagation within A in constant time, and finally obtain, at the counter of the final state of A, the new value for the target node of the edge labeled a_A in the top-level NFA. Therefore, all the edges (normal and special) of the top-level NFA can be traversed in constant time, so the costs at G_i can be obtained in $O(mn/r)$ time using Theorem 1. Now one propagates the costs to G_{i+1}, using the four-Russians tables to obtain the current counter values of each subgraph A in G_{i+1}.

Theorem 2 (Wu et al. 1995 [10]) *There exists an $O(n + mn/\log_{k+2} n)$ worst-case time solution to the AREM problem under weighted edit distance if the weights are integer numbers.*

Applications

The problem has applications in computational biology, to find certain types of motifs in DNA and protein sequences. See [1] for a more detailed discussion. In particular, PROSITE patterns are limited regular expressions rather popular to search protein sequences. PROSITE patterns can be searched for with faster algorithms in practice [7]. The same occurs with other classes of complex patterns [6] and network expressions [2].

Open Problems

The worst-case complexity of the AREM problem is not fully understood. It is of course $\Omega(n)$, which has been achieved for $m\log(k+2) = O(\log n)$, but it is not known how much can this be improved.

Experimental Results

Some recent experiments are reported in [5]. For small m and k, and assuming all the weights are 1 (except $w(c \to c) = 0$), bit-parallel algorithms of worst-case complexity $O(kn(m/\log n)^2)$ [4,9] are the fastest (the second is able to skip some text characters, depending on R). For arbitrary integer weights, the best choice is a more complex bit-parallel algorithm [5]; or the four-Russians based one [10] for larger m and k. The original algorithm [3] is slower but it is the only one supporting arbitrary weights.

URL to Code

Well-known packages offering efficient AREM (for simplified weight choices) are *agrep* [9] (http://webglimpse.net/download.html, top-level subdirectory `agrep/`) and *nrgrep* [4] (http://www.dcc.uchile.cl/~gnavarro/software). For biological applications, *anrep* [2] (http://www.cs.arizona.edu/people/gene/CODE/anrep.tar.Z) matches sequences of approximate network expressions with arbitrary weights and a specified gap length between each network expression and the next.

Cross References

- ► Regular Expression Matching is the simplified case where exact matching with strings in $\mathcal{L}(R)$ is sought.
- ► Sequential Approximate String Matching is a simplification of this problem, and the relation between graph G here and matrix C there should be apparent.

Recommended Reading

1. Gusfield, D.: Algorithms on strings, trees and sequences. Cambridge University Press, Cambridge (1997)
2. Myers, E.W.: Approximate matching of network expressions with spacers. J. Comput. Biol. **3**(1), 33–51 (1996)
3. Myers, E.W., Miller, W.: Approximate matching of regular expressions. Bullet. Math. Biol. **51**, 7–37 (1989)
4. Navarro, G.: Nr-grep: a fast and flexible pattern matching tool. Softw. Pr. Exp. **31**, 1265–1312 (2001)
5. Navarro, G.: Approximate regular expression searching with arbitrary integer weights. Nord. J. Comput. **11**(4), 356–373 (2004)
6. Navarro, G., Raffinot, M.: Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences. Cambridge University Press, Cambridge (2002)
7. Navarro, G., Raffinot, M.: Fast and simple character classes and bounded gaps pattern matching, with applications to protein searching. J. Comput. Biol. **10**(6), 903–923 (2003)
8. Thompson, K.: Regular expression search algorithm. Commun. ACM **11**(6), 419–422 (1968)
9. Wu, S., Manber, U.: Fast text searching allowing errors. Commun. ACM **35**(10), 83–91 (1992)
10. Wu, S., Manber, U., Myers, E.W.: A subquadratic algorithm for approximate regular expression matching. J. Algorithms **19**(3), 346–360 (1995)

Approximate Repetitions

► Approximate Tandem Repeats

Approximate Tandem Repeats

2001; Landau, Schmidt, Sokol
2003; Kolpakov, Kucherov

Gregory Kucherov[1], Dina Sokol[2]

[1] LIFL and INRIA, Villeneuve d'Ascq, France

[2] Department of Computer and Information Science, Brooklyn College of CUNY, Brooklyn, NY, USA

Keywords and Synonyms

Approximate repetitions; Approximate periodicities

Problem Definition

Identification of periodic structures in words (variants of which are known as *tandem repeats, repetitions, powers* or *runs*) is a fundamental algorithmic task (see entry ► Squares and Repetitions). In many practical applications, such as DNA sequence analysis, considered repetitions admit a certain variation between copies of the repeated pattern. In other words, repetitions under interest are *approximate tandem repeats* and not necessarily exact repeats only.

The simplest instance of an approximate tandem repeat is an *approximate square*. An approximate square in a word w is a subword uv, where u and v are within a given distance k according to some distance measure between words, such as Hamming distance or edit (also called Levenstein) distance. There are several ways to define approximate tandem repeats as successions of approximate squares, i. e. to generalize to the approximate case the notion of arbitrary periodicity (see entry ► Squares and Repetitions). In this entry, we discuss three different definitions of approximate tandem repeats. The first two are built upon the Hamming distance measure, and the third one is built upon the edit distance.

Let $h(\cdot,\cdot)$ denote the Hamming distance between two words of equal length.

Definition 1 A word $r[1..n]$ is called a K-*repetition* of period p, $p \leq n/2$, iff $h(r[1..n-p], r[p+1..n]) \leq K$.

Equivalently, a word $r[1..n]$ is a K-repetition of period p, if the number of mismatches, i. e. the number of i such that $r[i] \neq r[i+p]$, is at most K. For example, *ataa atta ctta ct* is a 2-repetition of period 4. *atc atc atc atg atg atg atg atg* is a 1-repetition of period 3 but *atc atc atc att atc atc atc att* is not.

Definition 2 A word $r[1..n]$ is called a K-*run*, of period p, $p \leq n/2$, iff for every $i \in [1..n-2p+1]$, we have $h(r[i..i+p-1], r[i+p, i+2p-1]) \leq K$.

A K-run can be seen as a sequence of approximate squares uv such that $|u| = |v| = p$ and u and v differ by at most K mismatches. The total number of mismatches in a K-run is not bounded.

Let $ed(\cdot,\cdot)$ denote the edit distance between two strings.

Definition 3 A word r is a K-*edit* repeat if it can be partitioned into consecutive subwords, $r = v'w_1w_2\ldots w_\ell v''$,

$\ell \geq 2$, such that

$$ed(v', w'_1) + \sum_{i=1}^{\ell-1} ed(w_i, w_{i+1}) + ed(w''_\ell, v'') \leq K\,,$$

where w'_1 is some suffix of w_1 and w''_ℓ is some prefix of w_ℓ. A K-edit repeat is a sequence of "evolving" copies of a pattern such that there are at most K insertions, deletions, and mismatches, overall, between all consecutive copies of the repeat. For example, the word $r = caagct\,cagct\,ccgct$ is a 2-edit repeat.

When looking for tandem repeats occurring in a word, it is natural to consider *maximal* repeats. Those are the repeats extended to the right and left as much as possible provided that the corresponding definition is still verified. Note that the notion of maximality applies to K-repetitions, to K-runs, and to K-edit repeats.

Under the Hamming distance, K-runs provide the weakest "reasonable" definition of approximate tandem repeats, since it requires that every square it contains cannot contain more than K mismatch errors, which seems to be a minimal reasonable requirement. On the other hand, K-repetition is the strongest such notion as it limits by K the *total* number of mismatches. This provides an additional justification that finding these two types of repeats is important as they "embrace" other intermediate types of repeats. Several intermediate definitions have been discussed in [10, Section 5].

In general, each K-repetition is a part of a K-run of the same period and every K-run is the union of all K-repetitions it contains. Observe that a K-run can contain as many as a linear number of K-repetitions with the same period. For example, the word $(000\,100)^n$ of length $6n$ is a 1-run of period 3, which contains $(2n-1)$ 1-repetitions. In general, a K-run r contains $(s - K + 1)$ K-repetitions of the same period, where s is the number of mismatches in r.

Example 1 The following Fibonacci word contains three 3-runs of period 6. They are shown in regular font, in positions aligned with their occurrences. Two of them are identical, and contain each four 3-repetitions, shown in italic for the first run only. The third run is a 3-repetition in itself.

```
010010   100100   101001   010010   010100   1001

 10010   100100   101001
 10010   100100   10
  0010   100100   101
    10   100100   10100
     0   100100   101001
                    1001   010010   010100   1
                             10   010100   1001
```

Key Results

Given a word w of length n and an integer K, it is possible to find all K-runs, K-repetitions, and K-edit repeats within w in the following time and space bounds:

***K*-runs** can be found in time $O(nK \log K + S)$ (S the output size) and working space $O(n)$ [10],

***K*-repetitions** can be found in time $O(nK \log K + S)$ and working space $O(n)$ [10],

***K*-edit repeats** can be found in time $O(nK \log K \log(n/K) + S)$ and working space $O(n + K^2)$ [14,19].

All three algorithms are based on similar algorithmic tools that generalize corresponding techniques for the exact case [4,15,16] (see [11] for a systematic presentation). The first basic tool is a generalization of *longest extension functions* [16] that, in the case of Hamming distance, can be exemplified as follows. Given a word w, we want to compute, for each position p and each $k \leq K$, the quantity $\max\{j | h(w[1..j], w[p..p+j-1]) \leq k\}$. Computing all those values can be done in time $O(nK)$ using a method based on the suffix tree and the computation of *lowest common ancestor* described in [7].

The second tool is the Lempel–Ziv factorization used in the well-known compression method. Different variants of the Lempel–Ziv factorization of a word can be computed in linear time [7,18].

The algorithm for computing K-repetitions from [10] can be seen as a direct generalization of the algorithm for computing maximal repetitions (runs) in the exact case [8,15]. Although based on the same basic tools and ideas, the algorithm [10] for computing K-runs is much more involved and uses a complex "bootstrapping" technique for assembling runs from smaller parts.

The algorithm for finding the K-edit repeats uses both the recursive framework and the idea of the *longest extension functions* of [16]. The longest common extensions, in this case, allow up to K edit operations. Efficient methods for computing these extensions are based upon a combination of the results of [12] and [13]. The K-edit repeats are derived by combining the longest common extensions computed in the forward direction with those computed in the reverse direction.

Applications

Tandemly repeated patterns in DNA sequences are involved in various biological functions and are used in different practical applications.

Tandem repeats are known to be involved in regulatory mechanisms, e. g. to act as binding sites for regulatory proteins. Tandem repeats have been shown to be associated with recombination hot-spots in higher organisms. In bacteria, a correlation has been observed between certain tandem repeats and virulence and pathogenicity genes.

Tandem repeats are responsible for a number of inherited diseases, especially those involving the central nervous system. Fragile X syndrome, Kennedy disease, myotonic dystrophy, and Huntington's disease are among the diseases that have been associated with triplet repeats.

Examples of different genetic studies illustrating above-mentioned biological roles of tandem repeats can be found in introductive sections of [1,6,9]. Even more than just genomic elements associated with various biological functions, tandem repeats have been established to be a fundamental mutational mechanism in genome evolution [17].

A major practical application of short tandem repeats is based on the inter-individual variability in copy number of certain repeats occurring at a single loci. This feature makes tandem repeats a convenient tool for genetic profiling of individuals. The latter, in turn, is applied to pedigree analysis and establishing phylogenetic relationships between species, as well as to forensic medicine [3].

Open Problems

The definition of K-edit repeats is similar to that of K-repetitions (for the Hamming distance case). It would be interesting to consider other definitions of maximal repeats over the edit distance. For example, a definition similar to the K-run would allow up to K edits between each pair of neighboring periods in the repeat. Other possible definitions would allow K errors between *any* pair of copies of a repeat, or between *all pairs* of copies, or between some *consensus* and each copy.

In general, a *weighted* edit distance scheme is necessary for biological applications. Known algorithms for tandem repeats based on a weighted edit distance scheme are not feasible, and thus only heuristics are currently used.

URL to Code

The algorithms described in this entry have been implemented for DNA sequences, and are publicly available. The Hamming distance algorithms (K-runs and K-repetitions) are part of the `mreps` software package, available at http://bioinfo.lifl.fr/mreps/ [9]. The K-edit repeats software, *TRED*, is available at http://www.sci.brooklyn.cuny.edu/~sokol/tandem [19]. The implementations of the algorithms are coupled with postprocessing filters, necessary due to the nature of biological sequences.

In practice, software based on heuristic and statistical methods is largely used. Among them, TRF (http://tandem.bu.edu/trf/trf.html) [1] is the most popular program used by the bioinformatics community. Other programs include ATRHunter (http://bioinfo.cs.technion.ac.il/atrhunter/) [20], TandemSWAN (http://strand.imb.ac.ru/swan/) [2]. STAR (http://atgc.lirmm.fr/star/) [5] is another software, based on an information-theoretic approach, for computing approximate tandem repeats of a pre-specified pattern.

Cross References

► Squares and Repetitions

Acknowledgments

This work was supported in part by the National Science Foundation Grant DB&I 0542751.

Recommended Reading

1. Benson, G.: Tandem Repeats Finder: a program to analyze DNA sequences. Nucleic Acids Res. **27**, 573–580 (1999)
2. Boeva, V.A., Régnier, M., Makeev, V.J.: SWAN: searching for highly divergent tandem repeats in DNA sequences with the evaluation of their statistical significance. Proceedings of JOBIM 2004, Montreal, Canada, p. 40 (2004)
3. Butler, J.M.: Forensic DNA Typing: Biology and Technology Behind STR Markers. Academic Press (2001)
4. Crochemore, M.: Recherche linéaire d'un carré dans un mot. Comptes Rendus Acad. Sci. Paris Sér. I Math. **296**, 781–784 (1983)
5. Delgrange, O., Rivals, E.: STAR – an algorithm to Search for Tandem Approximate Repeats. Bioinform. **20**, 2812–2820 (2004)
6. Gelfand, Y., Rodriguez, A., Benson, G.: TRDB – The Tandem Repeats Database. Nucl. Acids Res. **35**(suppl. 1), D80–D87 (2007)
7. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press (1997)
8. Kolpakov, R., Kucherov, G.: Finding maximal repetitions in a word in linear time. In: 40th Symp. Foundations of Computer Science (FOCS), pp. 596–604. IEEE Computer Society Press (1999)
9. Kolpakov, R., Bana, G., Kucherov, G.: `mreps`: efficient and flexible detection of tandem repeats in DNA. Nucl. Acids Res. **31**(13), 3672–3678 (2003)
10. Kolpakov, R., Kucherov, G.: Finding approximate repetitions under Hamming distance. Theoret. Comput. Sci. **33**(1), 135–156, (2003)
11. Kolpakov, R., Kucherov, G.: Identification of periodic structures in words. In: Berstel, J., Perrin, D. (eds.) Applied combinatorics on words. Encyclopedia of Mathematics and its Applications. Lothaire books, vol. 104, pp. 430–477. Cambridge University Press (2005)
12. Landau, G.M., Vishkin, U.: Fast string matching with k differences. J. Comput. Syst. Sci. **37**(1), 63–78 (1988)
13. Landau, G.M., Myers, E.W., Schmidt, J.P.: Incremental string comparison. SIAM J. Comput. **27**(2), 557–582 (1998)

14. Landau, G.M., Schmidt, J.P., Sokol, D.: An algorithm for approximate tandem repeats. J. Comput. Biol. **8**, 1–18 (2001)
15. Main, M.: Detecting leftmost maximal periodicities. Discret. Appl. Math. **25**, 145–153 (1989)
16. Main, M., Lorentz, R.: An $O(n \log n)$ algorithm for finding all repetitions in a string. J. Algorithms **5**(3), 422–432 (1984)
17. Messer, P.W., Arndt, P.F.: The majority of recent short DNA insertions in the human genome are tandem duplications. Mol. Biol. Evol. **24**(5), 1190–7 (2007)
18. Rodeh, M., Pratt, V., Even, S.: Linear algorithm for data compression via string matching. J. Assoc. Comput. Mach. **28**(1), 16–24 (1981)
19. Sokol, D., Benson, G., Tojeira, J.: Tandem repeats over the edit distance. Bioinform. **23**(2), e30–e35 (2006)
20. Wexler, Y., Yakhini, Z., Kashi, Y., Geiger, D.: Finding approximate tandem repeats in genomic sequences. J. Comput. Biol. **12**(7), 928–42 (2005)

Approximating Metric Spaces by Tree Metrics

1996; Bartal, Fakcharoenphol, Rao, Talwar
2004; Bartal, Fakcharoenphol, Rao, Talwar

JITTAT FAKCHAROENPHOL[1], SATISH RAO[2], KUNAL TALWAR[3]

[1] Department of Computer Engineering, Kasetsart University, Bangkok, Thailand

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

[3] Microsoft Research, Silicon Valley Campus, Mountain View, CA, USA

Keywords and Synonyms

Embedding general metrics into tree metrics

Problem Definition

This problem is to construct a random tree metric that probabilistically approximates a given arbitrary metric well. A solution to this problem is useful as the first step for numerous approximation algorithms because usually solving problems on trees is easier than on general graphs. It also finds applications in on-line and distributed computation.

It is known that tree metrics approximate general metrics badly, e. g., given a cycle C_n with n nodes, any tree metric approximating this graph metric has distortion $\Omega(n)$ [17]. However, Karp [15] noticed that a random spanning tree of C_n approximates the distances between any two nodes in C_n well in expectation. Alon, Karp, Peleg, and West [1] then proved a bound of $\exp(O(\sqrt{\log n \log\log n}))$ on an average distortion for approximating any graph metric with its spanning tree.

Bartal [2] formally defined the notion of probabilistic approximation.

Notations

A graph $G = (V, E)$ with an assignment of non-negative weights to the edges of G defines a metric space (V, d_G) where for each pair $u, v \in V$, $d_G(u, v)$ is the shortest path distance between u and v in G. A metric (V, d) is a *tree metric* if there exists some tree $T = (V', E')$ such that $V \subseteq V'$ and for all $u, v \in V$, $d_T(u, v) = d(u, v)$. The metric (V, d) is also called a metric induced by T.

Given a metric (V, d), a distribution $\mathcal{D}$ over tree metrics over V α*-probabilistically approximates* d if every tree metric $d_T \in \mathcal{D}$, $d_T(u, v) \geq d(u, v)$ and $\mathrm{E}_{d_T \in \mathcal{D}}[d_T(u, v)] \leq \alpha \cdot d(u, v)$, for every $u, v \in V$. The quantity α is referred to as the *distortion* of the approximation.

Although the definition of probabilistic approximation uses a distribution $\mathcal{D}$ over tree metrics, one is interested in a procedure that constructs a random tree metric distributed according to $\mathcal{D}$, i. e., an algorithm that produces a random tree metric that probabilistically approximates a given metric. The problem can be formally stated as follows.

Problem (APPROX-TREE)

INPUT: *a metric* (V, d)

OUTPUT: *a tree metric* (V, d_T) *sampled from a distribution* $\mathcal{D}$ *over tree metrics that* α*-probabilistically approximates* (V, d).

Bartal then defined a class of tree metrics, called hierarchically well-separated trees (HST), as follows. A k*-hierarchically well-separated tree* (k-HST) is a rooted weighted tree satisfying two properties: the edge weight from any node to each of its children is the same, and the edge weights along any path from the root to a leaf are decreasing by a factor of at least k. These properties are important to many approximation algorithms.

Bartal showed that any metric on n points can be probabilistically approximated by a set of k-HST's with $O(\log^2 n)$ distortion, an improvement from $\exp(O(\sqrt{\log n \log\log n}))$ in [1]. Later Bartal [3], following the same approach as in Seymour's analysis on the Feedback Arc Set problem [18], improved the distortion down to $O(\log n \log\log n)$. Using a rounding procedure of Calinescu, Karloff, and Rabani [5], Fakcharoenphol, Rao, and Talwar [9] devised an algorithm that, in expectation, produces a tree with $O(\log n)$ distortion. This bound is tight up to a constant factor.

Key Results

A tree metric is closely related to graph decomposition. The randomized rounding procedure of Calinescu, Karloff, and Rabani [5] for the 0-extension problem decomposes a graph into pieces with bounded diameter, cutting each edge with probability proportional to its length and a ratio between the numbers of nodes at certain distances. Fakcharoenphol, Rao, and Talwar [9] used the CKR rounding procedure to decompose the graph recursively and obtained the following theorem.

Theorem 1 *Given an n-point metric (V, d), there exists a randomized algorithm, which runs in time $O(n^2)$, that samples a tree metric from the distribution $\mathcal{D}$ over tree metrics that $O(\log n)$-probabilistically approximates (V, d). The tree is also a 2-HST.*

The bound in Theorem 1 is tight, as Alon et al. [1] proved the bound of an $\Omega(\log n)$ distortion when (V, d) is induced by a grid graph. Also note that it is known (as folklore) that even embedding a line metric onto a 2-HST requires distortion $\Omega(\log n)$.

If the tree is required to be a k-HST, one can apply the result of Bartal, Charikar, and Raz [4] which states that any 2-HST can be $O(k/\log k)$-probabilistically approximated by k-HST, to obtain an expected distortion of $O(k \log n/\log k)$.

Finding a distribution of tree metrics that probabilistically approximates a given metric has a dual problem that is to find a single tree T with small average weighted stretch. More specifically, given weight c_{uv} on edges, find a tree metric d_T such that for all $u, v \in V d_T(u, v) \geq d(u, v)$ and $\sum_{u,v \in V} c_{uv} \cdot d_T(u, v) \leq \alpha \sum_{u,v \in V} c_{uv} \cdot d(u, v)$.

Charikar, Chekuri, Goel, Guha, and Plotkin [6] showed how to find a distribution of $O(n \log n)$ tree metrics that α-probabilistically approximates a given metric, provided that one can solve the dual problem. The algorithm in Theorem 1 can be derandomized by the method of conditional expectation to find the required tree metric with $\alpha = O(\log n)$. Another algorithm based on modified region growing techniques is presented in [9], and independently by Bartal.

Theorem 2 *Given an n-point metric (V, d), there exists a polynomial-time deterministic algorithm that finds a distribution $\mathcal{D}$ over $O(n \log n)$ tree metrics that $O(\log n)$-probabilistically approximates (V, d).*

Note that the tree output by the algorithm contains Steiner nodes, however Gupta [10] showed how to find another tree metric without Steiner nodes while preserving all distances within a constant factor.

Applications

Metric approximation by random trees has applications in on-line and distributed computation, since randomization works well against oblivious adversaries, and trees are easy to work with and maintain. Alon et al. [1] first used tree embedding to give a competitive algorithm for the k-server problem. Bartal [3] noted a few problems in his paper: metrical task system, distributed paging, distributed k-server problem, distributed queuing, and mobile user.

After the paper by Bartal in 1996, numerous applications in approximation algorithms have been found. Many approximation algorithms work for problems on tree metrics or HST metrics. By approximating general metrics with these metrics, one can turn them into algorithms for general metrics, while, usually, losing only a factor of $O(\log n)$ in the approximation factors. Sample problems are metric labeling, buy-at-bulk network design, and group Steiner trees. Recent applications include an approximation algorithm to the Unique Games [12], information network design [13], and oblivious network design [11].

The SIGACT News article [8] is a review of the metric approximation by tree metrics with more detailed discussion on developments and techniques. See also [3,9], for other applications.

Open Problems

Given a metric induced by a graph, some application, e. g., solving a certain class of linear systems, does not only require a tree metric, but a tree metric induced by a spanning tree of the graph. Elkin, Emek, Spielman, and Teng [7] gave an algorithm for finding a spanning tree with average distortion of $O(\log^2 n \log \log n)$. It remains open if this bound is tight.

Cross References

▶ Metrical Task Systems
▶ Sparse Graph Spanners

Recommended Reading

1. Alon, N., Karp, R.M., Peleg, D., West, D.: A graph-theoretic game and its application to the k-server problem. SIAM J. Comput. **24**, 78–100 (1995)
2. Bartal, Y.: Probabilistic approximation of metric spaces and its algorithmic applications. In: FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science, Washington, DC, USA, IEEE Computer Society, pp. 184–193 (1996)
3. Bartal, Y.: On approximating arbitrary metrices by tree metrics. In: STOC '98: Proceedings of the thirtieth annual ACM sympo-

sium on Theory of computing, pp. 161–168. ACM Press, New York (1998)

4. Bartal, Y., Charikar, M., Raz, D.: Approximating min-sum k-clustering in metric spaces. In: STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing, pp. 11–20. ACM Press, New York (2001)
5. Calinescu, G., Karloff, H., Rabani, Y.: Approximation algorithms for the 0-extension problem. In: SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 8–16. (2001)
6. Charikar, M., Chekuri, C., Goel, A., Guha, S.: Rounding via trees: deterministic approximation algorithms for group steiner trees and k-median. In: STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing, pp. 114–123. ACM Press, New York (1998)
7. Elkin, M., Emek, Y., Spielman, D.A., Teng, S.-H.: Lower-stretch spanning trees. In: STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, pp. 494–503. ACM Press, New York (2005)
8. Fakcharoenphol, J., Rao, S., Talwar, K.: Approximating metrics by tree metrics. SIGACT News **35**, 60–70 (2004)
9. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. J. Comput. Syst. Sci. **69**, 485–497 (2004)
10. Gupta, A.: Steiner points in tree metrics don't (really) help. In: SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 220–227. (2001)
11. Gupta, A., Hajiaghayi, M.T., Räcke, H.: Oblivious network design. In: SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pp. 970–979. ACM Press, New York (2006)
12. Gupta, A., Talwar, K.: Approximating unique games. In: SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, New York, NY, USA, pp. 99–106. ACM Press, New York (2006)
13. Hayrapetyan, A., Swamy, C., Tardos, É.: Network design for information networks. In: SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 933–942. (2005)
14. Indyk, P., Matousek, J.: Low-distortion embeddings of finite metric spaces. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry. CRC Press, Inc., Chap. 8 (2004), To appear
15. Karp, R.: A 2*k*-competitive algorithm for the circle. Manuscript (1989)
16. Matousek, J.: Lectures on Discrete Geometry. Springer, New York (2002)
17. Rabinovich, Y., Raz, R.: Lower bounds on the distortion of embedding finite metric spaces in graphs. Discret. Comput. Geom. **19**, 79–94 (1998)
18. Seymour, P.D.: Packing directed circuits fractionally. Combinatorica **15**, 281–288 (1995)

Approximation Algorithm

▶ Knapsack

Approximation Algorithm Design

▶ Steiner Trees

Approximation Algorithms

▶ Graph Bandwidth

Approximation Algorithms in Planar Graphs

▶ Approximation Schemes for Planar Graph Problems

Approximations of Bimatrix Nash Equilibria

2003; Lipton, Markakis, Mehta
2006; Daskalaskis, Mehta, Papadimitriou
2006; Kontogiannis, Panagopoulou, Spirakis

SPYROS KONTOGIANNIS[1],
PANAGIOTA PANAGOPOULOU[2], PAUL SPIRAKIS[3]

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

Keywords and Synonyms

ϵ-Nash equilibria; ϵ-Well-supported Nash equilibria

Problem Definition

Nash [14] introduced the concept of Nash equilibria in non-cooperative games and proved that any game possesses at least one such equilibrium. A well-known algorithm for computing a Nash equilibrium of a 2-player game is the Lemke-Howson algorithm [12], however it has exponential worst-case running time in the number of available pure strategies [16].

Recently, Daskalakis et al. [5] showed that the problem of computing a Nash equilibrium in a game with 4 or more players is PPAD-complete; this result was later extended to games with 3 players [8]. Eventually, Chen and Deng [3] proved that the problem is PPAD-complete for 2-player games as well.

This fact emerged the computation of *approximate* Nash equilibria. There are several versions of approximate Nash equilibria that have been defined in the literature; however the focus of this entry is on the notions of ϵ-Nash equilibrium and ϵ-well-supported Nash equilibrium. An ϵ-Nash equilibrium is a strategy profile such that no deviating player could achieve a payoff higher than the one that the specific profile gives her, plus ϵ. A stronger notion of approximate Nash equilibria is the *ϵ-well-supported Nash equilibria*; these are strategy profiles such that each player plays only approximately best-response pure strategies with non-zero probability.

Notation

For a $n \times 1$ vector $\mathbf{x}$ denote by $x_1, \ldots, x_n$ the components of $\mathbf{x}$ and by $\mathbf{x}^{\mathrm{T}}$ the transpose of $\mathbf{x}$. Denote by $\mathbf{e_i}$ the column vector with a 1 at the ith coordinate and 0 elsewhere. For an $n \times m$ matrix A, denote a_{ij} the element in the i-th row and j-th column of A. Let $\mathbb{P}^n$ be the set of all probability vectors in n dimensions: $\mathbb{P}^n = \left\{\mathbf{z} \in \mathbb{R}^n_{\geq 0} : \sum_{i=1}^n z_i = 1\right\}$.

Bimatrix Games

Bimatrix games [18] are a special case of 2-player games such that the payoff functions can be described by two real $n \times m$ matrices A and B. The n rows of A, B represent the *action set* of the first player (the *row player*) and the m columns represent the action set of the second player (the *column player*). Then, when the row player chooses action i and the column player chooses action j, the former gets payoff a_{ij} while the latter gets payoff b_{ij}. Based on this, bimatrix games are denoted by $\Gamma = \langle A, B\rangle$.

A *strategy* for a player is any probability distribution on her set of actions. Therefore, a strategy for the row player can be expressed as a probability vector $\mathbf{x} \in \mathbb{P}^n$ while a strategy for the column player can be expressed as a probability vector $\mathbf{y} \in \mathbb{P}^m$. Each extreme point $\mathbf{e_i} \in \mathbb{P}^n$ ($\mathbf{e_j} \in \mathbb{P}^m$) that corresponds to the strategy assigning probability 1 to the i-th row (j-th column) is called a *pure strategy* for the row (column) player. A *strategy profile* $(\mathbf{x}, \mathbf{y})$ is a combination of (mixed in general) strategies, one for each player. In a given strategy profile $(\mathbf{x}, \mathbf{y})$ the players get *expected payoffs* $\mathbf{x}^{\mathrm{T}} A\mathbf{y}$ (row player) and $\mathbf{x}^{\mathrm{T}} B\mathbf{y}$ (column player).

If both payoff matrices belong to $[0, 1]^{m \times n}$ then the game is called a $[0, 1]$-bimatrix (or else, *positively normalized*) game. The special case of bimatrix games in which all elements of the matrices belong to $\{0, 1\}$ is called a $\{0, 1\}$-bimatrix (or else, *win-lose*) game. A bimatrix game $\langle A, B\rangle$ is called *zero sum* if $B = -A$.

Approximate Nash Equilibria

Definition 1 (ϵ-Nash equilibrium) For any $\epsilon > 0$ a strategy profile $(\mathbf{x}, \mathbf{y})$ is an *ϵ-Nash equilibrium* for the $n \times m$ bimatrix game $\Gamma = \langle A, B\rangle$ if

1. For all pure strategies $i \in \{1, \ldots, n\}$ of the row player, $\mathbf{e_i}^{\mathrm{T}} A\mathbf{y} \leq \mathbf{x}^{\mathrm{T}} A\mathbf{y} + \epsilon$ and
2. For all pure strategies $j \in \{1, \ldots, m\}$ of the column player, $\mathbf{x}^{\mathrm{T}} B\mathbf{e_j} \leq \mathbf{x}^{\mathrm{T}} B\mathbf{y} + \epsilon$.

Definition 2 (ϵ-well-supported Nash equilibrium) For any $\epsilon > 0$ a strategy profile $(\mathbf{x}, \mathbf{y})$ is an *ϵ-well-supported Nash equilibrium* for the $n \times m$ bimatrix game $\Gamma = \langle A, B\rangle$ if

1. For all pure strategies $i \in \{1, \ldots, n\}$ of the row player,
$$x_i > 0 \Rightarrow \mathbf{e_i}^{\mathrm{T}} A\mathbf{y} \geq \mathbf{e_k}^{\mathrm{T}} A\mathbf{y} - \epsilon \quad \forall k \in \{1, \ldots, n\}$$
2. For all pure strategies $j \in \{1, \ldots, m\}$ of the column player,
$$y_j > 0 \Rightarrow \mathbf{x}^{\mathrm{T}} B\mathbf{e_j} \geq \mathbf{x}^{\mathrm{T}} B\mathbf{e_k} - \epsilon \quad \forall k \in \{1, \ldots, m\}\,.$$

Note that both notions of approximate equilibria are defined with respect to an additive error term ϵ. Although (exact) Nash equilibria are known not to be affected by any positive scaling, it is important to mention that approximate notions of Nash equilibria are indeed affected. Therefore, the commonly used assumption in the literature when referring to approximate Nash equilibria is that the bimatrix game is positively normalized, and this assumption is adopted in the present entry.

Key Results

The work of Althöfer [1] shows that, for *any* probability vector $\mathbf{p}$ there exists a probability vector $\hat{\mathbf{p}}$ with logarithmic supports, so that for a fixed matrix C, $\max_j \left|\mathbf{p}^{\mathrm{T}} C\mathbf{e_j} - \hat{\mathbf{p}}^{\mathrm{T}} C\mathbf{e_j}\right| \leq \epsilon$, for any constant $\epsilon > 0$. Exploiting this fact, the work of Lipton, Markakis and Mehta [13], shows that, for any bimatrix game and for any *constant* $\epsilon > 0$, there exists an ϵ-Nash equilibrium with only logarithmic support (in the number n of available pure strategies). Consider a bimatrix game $\Gamma = \langle A, B\rangle$ and let $(\mathbf{x}, \mathbf{y})$ be a Nash equilibrium for Γ. Fix a positive integer k and form a multiset S_1 by sampling k times from the set of pure strategies of the row player, independently at random according to the distribution $\mathbf{x}$. Similarly, form a multiset S_2 by sampling k times from set of pure strategies of the column player according to $\mathbf{y}$. Let $\hat{\mathbf{x}}$ be the mixed strategy for the row player that assigns probability $1/k$ to each member of S_1 and 0 to all other pure strategies, and let $\hat{\mathbf{y}}$

be the mixed strategy for the column player that assigns probability $1/k$ to each member of S_2 and 0 to all other pure strategies. Then $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ are called *k-uniform* [13] and the following holds:

Theorem 1 **([13])** *For any Nash equilibrium* $(\mathbf{x}, \mathbf{y})$ *of a* $n \times n$ *bimatrix game and for every* $\epsilon > 0$*, there exists, for every* $k \geq (12 \ln n)/\epsilon^2$*, a pair of k-uniform strategies* $\hat{\mathbf{x}}, \hat{\mathbf{y}}$ *such that* $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ *is an* ϵ*-Nash equilibrium.*

This result directly yields a quasi-polynomial ($n^{O(\ln n)}$) algorithm for computing such an approximate equilibrium. Moreover, as pointed out in [1], no algorithm that examines supports smaller than about $\ln n$ can achieve an approximation better than 1/4.

Theorem 2 **([4])** *The problem of computing a* $1/n^{\Theta(1)}$*-Nash equilibrium of a* $n \times n$ *bimatrix game is* PPAD*-complete.*

Theorem 2 asserts that, unless PPAD $\subseteq$ P, there exists no fully polynomial time approximation scheme for computing equilibria in bimatrix games. However, this does not rule out the existence of a polynomial approximation scheme for computing an ϵ-Nash equilibrium when ϵ is an absolute constant, or even when $\epsilon = \Theta\left(1/poly(\ln n)\right)$. Furthermore, as observed in [4], if the problem of finding an ϵ-Nash equilibrium were PPAD-complete when ϵ is an absolute constant, then, due to Theorem 1, all PPAD problems would be solved in quasi-polynomial time, which is unlikely to be the case.

Two concurrent and independent works [6,10] were the first to make progress in providing ϵ-Nash equilibria and ϵ-well-supported Nash equilibria for bimatrix games and some *constant* $0 < \epsilon < 1$. In particular, the work of Kontogiannis, Panagopoulou and Spirakis [10] proposes a simple linear-time algorithm for computing a 3/4-Nash equilibrium for any bimatrix game:

Theorem 3 **([10])** *Consider any* $n \times m$ *bimatrix game* $\Gamma = \langle A, B \rangle$ *and let* $a_{i_1, j_1} = \max_{i,j} a_{ij}$ *and* $b_{i_2, j_2} = \max_{i,j} b_{ij}$*. Then the pair of strategies* $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ *where* $\hat{x}_{i_1} = \hat{x}_{i_2} = \hat{y}_{j_1} = \hat{y}_{j_2} = 1/2$ *is a 3/4-Nash equilibrium for* Γ*.*

The above technique can be extended so as to obtain a parametrized, stronger approximation:

Theorem 4 **([10])** *Consider a* $n \times m$ *bimatrix game* $\Gamma = \langle A, B \rangle$*. Let* λ_1^* (λ_2^*) *be the minimum, among all Nash equilibria of* Γ*, expected payoff for the row (column) player and let* $\lambda = \max\{\lambda_1^*, \lambda_2^*\}$*. Then, there exists a* $(2 + \lambda)/4$*-Nash equilibrium that can be computed in time polynomial in n and m.*

The work of Daskalakis, Mehta and Papadimitriou [6] provides a simple algorithm for computing a 1/2-Nash equilibrium: Pick an arbitrary row for the row player, say row i. Let $j = \arg\max_{j'} b_{ij'}$. Let $k = \arg\max_{k'} a_{k'j}$. Thus, j is a best-response column for the column player to the row i, and k is a best-response row for the row player to the column j. Let $\hat{\mathbf{x}} = 1/2\mathbf{e_i} + 1/2\mathbf{e_k}$ and $\hat{\mathbf{y}} = \mathbf{e_j}$, i. e., the row player plays row i or row k with probability 1/2 each, while the column player plays column j with probability 1. Then:

Theorem 5 **([6])** *The strategy profile* $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ *is a 1/2-Nash equilibrium.*

A polynomial construction (based on Linear Programming) of a 0.38-Nash equilibrium is presented in [7].

For the more demanding notion of well-supported approximate Nash equilibrium, Daskalakis, Mehta and Papadimitriou [6] propose an algorithm, which, under a quite interesting and plausible graph theoretic conjecture, constructs in polynomial time a 5/6-well-supported Nash equilibrium. However, the status of this conjecture is still unknown. In [6] it is also shown how to transform a $[0, 1]$-bimatrix game to a $\{0, 1\}$-bimatrix game of the same size, so that each ϵ-well supported Nash equilibrium of the resulting game is $(1 + \epsilon)/2$-well supported Nash equilibrium of the original game.

The work of Kontogiannis and Spirakis [11] provides a polynomial algorithm that computes a 1/2-well-supported Nash equilibrium for arbitrary win-lose games. The idea behind this algorithm is to split evenly the divergence from a zero sum game between the two players and then solve this zero sum game in polynomial time (using its direct connection to Linear Programming). The computed Nash equilibrium of the zero sum game considered is indeed proved to be also a 1/2-well-supported Nash equilibrium for the initial win-lose game. Therefore:

Theorem 6 **([11])** *For any win-lose bimatrix game, there is a polynomial time constructable profile that is a 1/2-well-supported Nash equilibrium of the game.*

In the same work, Kontogiannis and Spirakis [11] parametrize the above methodology in order to apply it to arbitrary bimatrix games. This new technique leads to a weaker φ-well-supported Nash equilibrium for win-lose games, where $\phi = (\sqrt{5} - 1)/2$ is the golden ratio. Nevertheless, this parametrized technique extends nicely to a technique for arbitrary bimatrix games, which assures a 0.658-well-supported Nash equilibrium in polynomial time:

Theorem 7 **([11])** *For any bimatrix game, a* $\left(\sqrt{11}/2 - 1\right)$*-well-supported Nash equilibrium is constructable in polynomial time.*

Two very new results improved the approximation status of ϵ- Nash Equilibria:

Theorem 8 ([2]) *There is a polynomial time algorithm, based on Linear Programming, that provides an 0.36392-Nash Equilibrium.*

The second result below is the best till now:

Theorem 9 ([17]) *There exists a polynomial time algorithm, based on the stationary points of a natural optimization problem, that provides an 0.3393-Nash Equilibrium.*

Kannan and Theobald [9] investigate a hierarchy of bimatrix games $\langle A, B\rangle$ which results from restricting the rank of the matrix $A + B$ to be of fixed rank at most k. They propose a new model of ϵ-approximation for games of rank k and, using results from quadratic optimization, show that approximate Nash equilibria of constant rank games can be computed deterministically in time polynomial in $1/\epsilon$. Moreover, [9] provides a randomized approximation algorithm for certain quadratic optimization problems, which yields a randomized approximation algorithm for the Nash equilibrium problem. This randomized algorithm has similar time complexity as the deterministic one, but it has the possibility of finding an exact solution in polynomial time if a conjecture is valid. Finally, they present a polynomial time algorithm for *relative approximation* (with respect to the payoffs in an equilibrium) provided that the matrix $A + B$ has a nonnegative decomposition.

Applications

Non-cooperative game theory and its main solution concept, i. e. the Nash equilibrium, have been extensively used to understand the phenomena observed when decision-makers interact and have been applied in many diverse academic fields, such as biology, economics, sociology and artificial intelligence. Since however the computation of a Nash equilibrium is in general PPAD-complete, it is important to provide efficient algorithms for approximating a Nash equilibrium; the algorithms discussed in this entry are a first step towards this direction.

Cross References

► Complexity of Bimatrix Nash Equilibria
► General Equilibrium
► Non-approximability of Bimatrix Nash Equilibria

Recommended Reading

1. Althöfer, I.: On sparse approximations to randomized strategies and convex combinations. Linear Algebr. Appl. **199**, 339–355 (1994)
2. Bosse, H., Byrka, J., Markakis, E.: New Algorithms for Approximate Nash Equilibria in Bimatrix Games. In: LNCS Proceedings of the 3rd International Workshop on Internet and Network Economics (WINE 2007), San Diego, 12–14 December 2007
3. Chen, X., Deng, X.: Settling the complexity of 2-player Nash-equilibrium. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06). Berkeley, 21–24 October 2005
4. Chen, X., Deng, X., Teng, S.-H.: Computing Nash equilibria: Approximation and smoothed complexity. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), Berkeley, 21–24 October 2006
5. Daskalakis, C., Goldberg, P., Papadimitriou, C.: The complexity of computing a Nash equilibrium. In: Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC'06), pp. 71–78. Seattle, 21–23 May 2006
6. Daskalakis, C., Mehta, A., Papadimitriou, C.: A note on approximate Nash equilibria. In: Proceedings of the 2nd Workshop on Internet and Network Economics (WINE'06), pp. 297–306. Patras, 15–17 December 2006
7. Daskalakis, C., Mehta, A., Papadimitriou, C: Progress in approximate Nash equilibrium. In: Proceedings of the 8th ACM Conference on Electronic Commerce (EC07), San Diego, 11–15 June 2007
8. Daskalakis, C., Papadimitriou, C.: Three-player games are hard. In: Electronic Colloquium on Computational Complexity (ECCC) (2005)
9. Kannan, R., Theobald, T.: Games of fixed rank: A hierarchy of bimatrix games. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, New Orleans, 7–9 January 2007
10. Kontogiannis, S., Panagopoulou, P.N., Spirakis, P.G.: Polynomial algorithms for approximating Nash equilibria of bimatrix games. In: Proceedings of the 2nd Workshop on Internet and Network Economics (WINE'06), pp. 286–296. Patras, 15–17 December 2006
11. Kontogiannis, S., Spirakis, P.G.: Efficient Algorithms for Constant Well Supported Approximate Equilibria in Bimatrix Games. In: Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP'07, Track A: Algorithms and Complexity), Wroclaw, 9–13 July 2007
12. Lemke, C.E., Howson, J.T.: Equilibrium points of bimatrix games. J. Soc. Indust. Appl. Math. **12**, 413–423 (1964)
13. Lipton, R.J., Markakis, E., Mehta, A.: Playing large games using simple startegies. In: Proceedings of the 4th ACM Conference on Electronic Commerce (EC'03), pp. 36–41. San Diego, 9–13 June 2003
14. Nash, J.: Noncooperative games. Ann. Math. **54**, 289–295 (1951)
15. Papadimitriou, C.H.: On inefficient proofs of existence and complexity classes. In: Proceedings of the 4th Czechoslovakian Symposium on Combinatorics 1990, Prachatice (1991)
16. Savani, R., von Stengel, B.: Exponentially many steps for finding a nash equilibrium in a bimatrix game. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), pp. 258–267. Rome, 17–19 October 2004
17. Tsaknakis, H., Spirakis, P.: An Optimization Approach for Approximate Nash Equilibria. In: LNCS Proceedings of the 3rd International Workshop on Internet and Network Economics (WINE 2007), also in the Electronic Colloquium on Computational Complexity, (ECCC), TR07-067 (Revision), San Diego, 12–14 December 2007

18. von Neumann, J., Morgenstern, O.: Theory of Games and Economic Behavior. Princeton University Press, Princeton, NJ (1944)

Approximation Schemes for Bin Packing
1982; Karmarker, Karp

NIKHIL BANSAL
IBM Research, IBM, Yorktown Heights, NY, USA

Keywords and Synonyms

Cutting stock problem

Problem Definition

In the bin packing problem, the input consists of a collection of items specified by their sizes. There are also identical bins, which without loss of generality can be assumed to be of size 1, and the goal is to pack these items using the minimum possible number of bins.

Bin packing is a classic optimization problem, and hundreds of its variants have been defined and studied under various settings such as average case analysis, worst-case offline analysis, and worst-case online analysis. This note considers the most basic variant mentioned above under the offline model where all the items are given in advance. The problem is easily seen to be NP-hard by a reduction from the partition problem. In fact, this reduction implies that unless P = NP, it impossible to determine in polynomial time whether the items can be packed into two bins or whether they need three bins.

Notations

The input to the bin packing problem is a set of n items I specified by their sizes $s_1, \ldots, s_n$, where each s_i is a real number in the range $(0, 1]$. A subset of items $S \subseteq I$ can be packed feasibly in a bin if the total size of items in S is at most 1. The goal is to pack all items in I into the minimum number of bins. Let $\mathrm{OPT}(I)$ denote the value of the optimum solution and $\mathrm{Size}(I)$ the total size of all items in I. Clearly, $\mathrm{OPT}(I) \geq \lceil \mathrm{Size}(I) \rceil$.

Strictly speaking, the problem does not admit a polynomial-time algorithm with an approximation guarantee better than 3/2. Interestingly, however, this does not rule out an algorithm that requires, say, $\mathrm{OPT}(I) + 1$ bins (unlike other optimization problems, making several copies of a small hard instance to obtain a larger hard instance does not work for bin packing). It is more meaningful to consider approximation guarantees in an asymptotic sense. An algorithm is called an asymptotic ρ approximation if the number of bins required by it is $\rho \cdot \mathrm{OPT}(I) + O(1)$.

Key Results

During the 1960s and 1970s several algorithms with constant factor asymptotic and absolute approximation guarantees and very efficient running times were designed (see [1] for a survey). A breakthrough was achieved in 1981 by de la Vega and Lueker [3], who gave the first polynomial-time asymptotic approximation scheme.

Theorem 1 ([3]) *Given any arbitrary parameter $\epsilon > 0$, there is an algorithm that uses $(1+\epsilon)OPT(I) + O(1)$ bins to pack I. The running time of this algorithm is $O(n \log n) + (1/\epsilon)^{O(1/\epsilon)}$.*

The main insight of de la Vega and Lueker [3] was to give a technique for approximating the original instance by a simpler instance where large items have only $O(1)$ distinct sizes. Their idea was simple. First, it suffices to restrict attention to large items, say, with size greater than ε. These can be called I_b. Given an (almost) optimum packing of I_b, consider the solution obtained by greedily filling up the bins with remaining small items, opening new bins only if needed. Indeed, if no new bins are needed, then the solution is still almost optimum since the packing for I_b was almost optimum. If additional bins are needed, then each bin, except possibly one, must be filled to an extent $(1-\epsilon)$, which gives a packing using $\mathrm{Size}(I)/(1-\epsilon) + 1 \leq \mathrm{OPT}(I)/(1-\epsilon) + 1$ bins. So it suffices to focus on solving I_b almost optimally. To do this, the authors show how to obtain another instance I' with the following properties. First, I' has only $O(1/\epsilon^2)$ distinct sizes, and second, I' is an approximation of I_b in the sense that $\mathrm{OPT}(I_b) \geq \mathrm{OPT}(I')$ and, moreover, any solution of I' implies another solution of I_b using $O(\epsilon \cdot \mathrm{OPT}(I))$ additional bins. As I' has only $1/\epsilon^2$ distinct item sizes, and any bin can obtain at most $1/\epsilon$ such items, there are at most $O(1/\epsilon^2)^{1/\epsilon}$ ways to pack a bin. Thus, I' can be solved optimally by exhaustive enumeration (or more efficiently using an integer programming formulation described below).

Later, Karmarkar and Karp [4] proved a substantially stronger guarantee.

Theorem 2 ([4]) *Given an instance I, there is an algorithm that produces a packing of I using $OPT(I) + O(\log^2 OPT(I))$ bins. The running time of this algorithm is $O(n^8)$.*

Observe that this guarantee is significantly stronger than that of [3] as the additive term is $O(\log^2 \mathrm{OPT})$ as opposed to $O(\epsilon \cdot \mathrm{OPT})$. Their algorithm also uses the ideas of reducing the number of distinct item sizes and ignoring small items, but in a much more refined way. In particular, instead of obtaining a rounded instance in a single step, their algorithm consists of a logarithmic number of steps where in each step they round the instance "mildly" and then solve it partially.

The starting point is an exponentially large linear programming (LP) relaxation of the problem commonly referred to as the configuration LP. Here there is a variable x_S corresponding to each subset of items S that can be packed feasibly in a bin. The objective is to minimize $\sum_S x_S$ subject to the constraint that for each item i, the sum of x_S over all subsets S that contain i is at least 1. Clearly, this is a relaxation as setting $x_S = 1$ for each set S corresponding to a bin in the optimum solution is a feasible integral solution to the LP. Even though this formulation has exponential size, the separation problem for the dual is a knapsack problem, and hence the LP can be solved in polynomial time to any accuracy (in particular within an accuracy of 1) using the ellipsoid method. Such a solution is called a fractional packing. Observe that if there are n_i items each of size exactly s_i, then the constraints corresponding to i can be "combined" to obtain the following LP:

$$\begin{aligned} &\min \sum_S x_S \\ \text{s.t.}\quad &\sum_S a_{S,i} x_S \geq n_i \qquad &&\forall \text{ item sizes } i \\ &x_S \geq 0 &&\forall \text{ feasible sets } S. \end{aligned}$$

Here $a_{S,i}$ is the number of items of size s_i in the feasible S. Let $q(I)$ denote the number of distinct sizes in I. The number of nontrivial constraints in LP is equal to $q(I)$, which implies that there is a basic optimal solution to this LP that has only $q(I)$ variables set nonintegrally. Karmarkar and Karp exploit this observation in a very clever way. The following lemma describes the main idea.

Lemma 3 *Given any instance J, suppose there is an algorithmic rounding procedure to obtain another instance J' such that J' has Size$(J)/2$ distinct item sizes and J and J' are related in the following sense: given any fractional packing of J using ℓ bins gives a fractional packing of J' with at most ℓ bins, and given any packing of J' using ℓ' bins gives a packing of J using $\ell' + c$ bins, where c is some fixed parameter. Then J can be packed using OPT$(J) + c \cdot \log(OPT(J))$ bins.*

Proof Let $I_0 = I$ and let I_1 be the instance obtained by applying the rounding procedure to I_0. By the property of the rounding procedure, $\mathrm{OPT}(I) \leq \mathrm{OPT}(I_1) + c$ and $\mathrm{LP}(I_1) \leq \mathrm{LP}(I)$. As I_1 has $\mathrm{Size}(I_0)/2$ distinct sizes, the LP solution for I_1 has at most $\mathrm{Size}(I_0)/2$ fractionally set variables. Remove the items packed integrally in the LP solution and consider the residual instance I'_1. Note that $\mathrm{Size}(I'_1) \leq \mathrm{Size}(I_0)/2$. Now, again apply the rounding procedure to I'_1 to obtain I_2 and solve the LP for I_2. Again, this solution has at most $\mathrm{Size}(I'_1)/2 \leq \mathrm{Size}(I_0)/4$ fractionally set variables, and $\mathrm{OPT}(I'_1) \leq \mathrm{OPT}(I_2) + c$ and $\mathrm{LP}(I_2) \leq \mathrm{LP}(I'_1)$. The above process is repeated for a few steps. At each step, the size of the residual instance decreases by a factor of at least two, and the number of bins required to pack I_0 increases by additive c. After $\log(\mathrm{Size}(I_0))$ ($\approx \log(\mathrm{OPT}(I))$) steps, the residual instance has size $O(1)$ and can be packed into $O(1)$ additional bins. □

It remains to describe the rounding procedure. Consider the items in nondecreasing order $s_1 \geq s_2 \geq \ldots \geq s_n$ and group them as follows. Add items to current group until its size first exceeds 2. At this point close the group and start a new group. Let $G_1, \ldots, G_k$ denote the groups formed and let $n_i = |G_i|$, setting $n_0 = 0$ for convenience. Define I' as the instance obtained by rounding the size of $n_{\mathrm{i}-1}$ largest items in G_i to the size of the largest item in G_i for $i = 1, \ldots, k$. The procedure satisfies the properties of Lemma 3 with $c = O(\log n_k)$ (left as an exercise to the reader). To prove Theorem 2, it suffices to show that $n_k = O(\mathrm{Size}(I))$. This is done easily by ignoring all items smaller than $1/\mathrm{Size}(I)$ and filling them in only in the end (as in the algorithm of de la Vega and Lueker).

In the case when the item sizes are not too small, the following corollary is obtained.

Corollary 1 *If all the item sizes are at least δ, it is easily seen that $c = O(\log 1/\delta)$, and the above algorithm implies a guarantee of OPT $+ O(\log(1/\delta) \cdot \log$ OPT$)$, which is OPT $+ O(\log$ OPT$)$ if δ is a constant.*

Applications

The bin packing problem is directly motivated from practice and has many natural applications such as packing items into boxes subject to weight constraints, packing files into CDs, packing television commercials into station breaks, and so on. It is widely studied in operations research and computer science. Other applications include the so-called cutting-stock problems where some material such as cloth or lumber is given in blocks of standard size from which items of certain specified size must be cut. Several variations of bin packing, such as generalizations to higher dimensions, imposing additional constraints on the algorithm and different optimization criteria, have also been extensively studied. The reader is referred to [1,2] for excellent surveys.

Open Problems

Except for the NP-hardness, no other hardness results are known and it is possible that a polynomial-time algorithm with guarantee OPT + 1 exists for the problem. Resolving this is a key open question. A promising approach seems to be via the configuration LP (considered above). In fact, no instance is known for which the additive gap between the optimum configuration LP solution and the optimum integral solution is more than 1. It would be very interesting to design an instance that has an additive integrality gap of two or more.

The OPT + $O(\log^2 \text{OPT})$ guarantee of Karmarkar and Karp has been the best known result for the last 25 years, and any improvement to this would be an extremely interesting result by itself.

Cross References

▶ Bin Packing
▶ Knapsack

Recommended Reading

1. Coffman, E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for bin packing: a survey. In: Hochbaum, D. (ed.) Approximation Algorithms for NP-hard Problems, pp. 46–93. PWS, Boston (1996)
2. Csirik, J., Woeginger, G.: On-line packing and covering problems. In: Fiat, A., Woeginger, G. (eds.) Online Algorithms: The State of the Art. LNCS, vol. 1442, pp. 147–177. Springer, Berlin (1998)
3. Fernandez de la Vega, W., Lueker, G.: Bin packing can be solved within $1 + \varepsilon$ in linear time. Combinatorica **1**, 349–355 (1981)
4. Karmarkar, N., Karp, R.M.: An efficient approximation scheme for the one-dimensional bin-packing problem. In: Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science (FOCS), 1982, pp. 312–320

Approximation Schemes for Planar Graph Problems

1983; Baker
1994; Baker

Erik D. Demaine[1], MohammadTaghi Hajiaghayi[2]
[1] Computer Science and Artifical Intelligence Laboratory, MIT, Cambridge, MA, USA
[2] Department of Computer Science,
University of Pittsburgh,
Pittsburgh, PA, USA

Keywords and Synonyms

Approximation algorithms in planar graphs; Baker's approach; Lipton–Tarjan approach

Problem Definition

Many NP-hard graph problems become easier to approximate on planar graphs and their generalizations. (A graph is *planar* if it can be drawn in the plane (or the sphere) without crossings. For definitions of other related graph classes, see the entry on ▶ bidimensionality (2004; Demaine, Fomin, Hajiaghayi, Thilikos).) For example, *maximum independent set* asks to find a maximum subset of vertices in a graph that induce no edges. This problem is inapproximable in general graphs within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$ unless NP = ZPP (and inapproximable within $n^{1/2-\epsilon}$ unless P = NP), while for planar graphs there is a 4-approximation (or simple 5-approximation) by taking the largest color class in a vertex 4-coloring (or 5-coloring). Another is *minimum dominating set*, where the goal is to find a minimum subset of vertices such that every vertex is either in or adjacent to the subset. This problem is inapproximable in general graphs within $\epsilon \log n$ for some $\epsilon > 0$ unless P = NP, but as we will see, for planar graphs the problem admits a *polynomial-time approximation scheme* (PTAS): a collection of $(1 + \epsilon)$-approximation algorithms for all $\epsilon > 0$.

There are two main general approaches to designing PTASs for problems on planar graphs and their generalizations: the separator approach and the Baker approach.

Lipton and Tarjan [15,16] introduced the first approach, which is based on planar separators. The first step in this approach is to find a separator of $O(\sqrt{n})$ vertices or edges, where n is the size of the graph, whose removal splits the graph into two or more pieces each of which is a constant fraction smaller than the original graph. Then recurse in each piece, building a recursion tree of separators, and stop when the pieces have some constant size such as $1/\epsilon$. The problem can be solved on these pieces by brute force, and then it remains to combine the solutions up the recursion tree. The induced error can often be bounded in terms of the total size of all separators, which in turn can be bounded by $\epsilon\, n$. If the optimal solution is at least some constant factor times n, this approach often leads to a PTAS.

There are two limitations to this planar-separator approach. First, it requires that the optimal solution be at least some constant factor times n; otherwise, the cost incurred by the separators can be far larger than the desired optimal solution. Such a bound is possible in some problems after some graph pruning (linear kernelization), e. g., independent set, vertex cover, and forms of the traveling salesman problem. But, for example, Grohe [12] states that the dominating set is a problem "to which the technique based on the separator theorem does not apply." Second, the approximation algorithms resulting from planar separators are often impractical because of large constant factors. For example, to achieve an approximation ratio of just 2, the base case requires exhaustive solution of graphs of up to $2^{2^{400}}$ vertices.

Baker [1] introduced her approach to address the second limitation, but it also addresses the first limitation to a certain extent. This approach is based on decomposition into overlapping subgraphs of bounded outerplanarity, as described in the next section.

Key Results

Baker's original result [1] is a PTAS for a maximum independent set (as defined above) on planar graphs, as well as the following list of problems on planar graphs: maximum tile salvage, partition into triangles, maximum H-matching, minimum vertex cover, minimum dominating set, and minimum edge-dominating set.

Baker's approach starts with a planar embedding of the planar graph. Then it divides vertices into *layers* by iteratively removing vertices on the outer face of the graph: layer j consists of the vertices removed at the jth iteration. If one now removes the layers congruent to i modulo k, for any choice of i, the graph separates into connected components each with at most k consecutive layers, and hence the graph becomes k-outerplanar. Many NP-complete problems can be solved on k-outerplanar graphs for fixed k using dynamic programming (in particular, such graphs have bounded treewidth). Baker's approximation algorithm computes these optimal solutions for each choice i of the congruence class of layers to remove and returns the best solution among these k solutions. The key argument for maximization problems considers the optimal solution to the full graph and argues that the removal of one of the k congruence classes of layers must remove at most a $1/k$ fraction of the optimal solution, so the returned solution must be within a $1 + 1/k$ factor of optimal. A more delicate argument handles minimization problems as well. For many problems, such as maximum independent set, minimum dominating set, and minimum vertex cover, Baker's approach obtains a $(1 + \epsilon)$-approximation algorithms with a running time of $2^{O(1/\epsilon)}n^{O(1)}$ on planar graphs.

Eppstein [10] generalized Baker's approach to a broader class of graphs called graphs of *bounded local treewidth*, i. e., where the treewidth of the subgraph induced by the set of vertices at a distance of at most r from any vertex is bounded above by some function $f(r)$ independent of n. The main differences in Eppstein's approach are replacing the concept of bounded outerplanarity with the concept of bounded treewidth, where dynamic programming can still solve many problems, and labeling layers according to a simple breadth-first search. This approach has led to PTASs for hereditary maximization problems such as maximum independent set and maximum clique, maximum triangle matching, maximum H-matching, maximum tile salvage, minimum vertex cover, minimum dominating set, minimum edge-dominating set, minimum color sum, and subgraph isomorphism for a fixed pattern [6,8,10]. Frick and Grohe [11] also developed a general framework for deciding any property expressible in first-order logic in graphs of bounded local treewidth.

The foundation of these results is Eppstein's characterization of minor-closed families of graphs with bounded local treewidth [10]. Specifically, he showed that a minor-closed family has bounded local treewidth if and only if it excludes some *apex graph*, a graph with a vertex whose removal leaves a planar graph. Unfortunately, the initial proof of this result brought Eppstein's approach back to the realm of impracticality, because his bound on local treewidth in a general apex-minor-free graph is doubly exponential in r: $2^{2^{O(r)}}$. Fortunately, this bound could be improved to $2^{O(r)}$ [3] and even the optimal $O(r)$ [4]. The latter bound restores Baker's $2^{O(1/\epsilon)}n^{O(1)}$ running time for $(1 + \epsilon)$-approximation algorithms, now for all apex-minor-free graphs.

Another way to view the necessary decomposition of Baker's and Eppstein's approaches is that the vertices or edges of the graph can be split into any number k of pieces such that deleting any one of the pieces results in a graph of bounded treewidth (where the bound depends on k). Such decompositions in fact exist for arbitrary graphs excluding any fixed minor H [9], and they can be found in polynomial time [6]. This approach generalizes the Baker–Eppstein PTASs described above to handle general H-minor-free graphs.

This decomposition approach is effectively limited to *deletion-closed* problems, whose optimal solution only improves when deleting edges or vertices from the graph. Another decomposition approach targets *contraction-closed* problems, whose optimal solution only improves when contracting edges. These problems include classic problems such as dominating set and its variations, the traveling salesman problem, subset TSP, minimum Steiner tree, and minimum-weight c-edge-connected submultigraph. PTASs have been obtained for these problems in planar graphs [2,13,14] and in bounded-genus graphs [7] by showing that the edges can be decomposed into any number k of pieces such that contracting any one piece results in a bounded-treewidth graph (where the bound depends on k).

Applications

Most applications of Baker's approach have been limited to optimization problems arising from "local" properties (such as those definable in first-order logic). Intuitively, such local properties can be decided by locally checking every constant-size neighborhood. In [5], Baker's approach is generalized to obtain PTASs for nonlocal problems, in particular, connected dominating set. This generalization requires the use of two different techniques. The first technique is to use an ε-fraction of a constant-factor (or even logarithmic-factor) approximation to the problem as a "backbone" for achieving the needed nonlocal property. The second technique is to use subproblems that overlap by $\Theta(\log n)$ layers instead of the usual $\Theta(1)$ in Baker's approach.

Despite this advance in applying Baker's approach to more general problems, the planar-separator approach can still handle some different problems. Recall, though, that the planar-separator approach was limited to problems in which the optimal solution is at least some constant factor times n. This limitation has been overcome for a wide range of problems [5], in particular obtaining a PTAS for feedback vertex set, to which neither Baker's approach nor the planar-separator approach could previously apply. This result is based on evenly dividing the optimum solution instead of the whole graph, using a relation between treewidth and the optimal solution value to bound the treewidth of the graph, and thus obtaining an $O(\sqrt{\mathrm{OPT}})$ separator instead of an $O(\sqrt{n})$ separator. The $O(\sqrt{\mathrm{OPT}})$ bound on treewidth follows from the bidimensionality theory described in the entry on ▶ bidimensionality (2004; Demaine, Fomin, Hajiaghayi, Thilikos). We can divide the optimum solution into roughly even pieces, without knowing the optimum solution, by using existing constant-factor (or even logarithmic-factor) approximations for the problem. At the base of the recursion, pieces no longer have bounded size but do have bounded treewidth, so fast fixed-parameter algorithms can be used to construct optimal solutions.

Open Problems

An intriguing direction for future research is to build a general theory for PTASs of subset problems. Although PTASs for subset TSP and Steiner tree have recently been obtained for planar graphs [2,14], there remain several open problems of this kind, such as subset feedback vertex set.

Another instructive problem is to understand the extent to which Baker's approach can be applied to nonlocal problems. Again there is an example of how to modify the approach to handle the nonlocal problem of connected dominating set [5], but for example the only known PTAS for feedback vertex set in planar graphs follows the separator approach.

Cross References

▶ Bidimensionality
▶ Separators in Graphs
▶ Treewidth of Graphs

Recommended Reading

1. Baker, B.S.: Approximation algorithms for NP-complete problems on planar graphs. J. Assoc. Comput. Mach. **41**(1), 153–180 (1994)
2. Borradaile, G., Kenyon-Mathieu, C., Klein, P.N.: A polynomial-time approximation scheme for Steiner tree in planar graphs. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, 2007
3. Demaine, E.D., Hajiaghayi, M.: Diameter and treewidth in minor-closed graph families, revisited. Algorithmica **40**(3), 211–215 (2004)
4. Demaine, E.D., Hajiaghayi, M.: Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In: Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA'04), January 2004, pp. 833–842
5. Demaine, E.D., Hajiaghayi, M.: Bidimensionality: new connections between FPT algorithms and PTASs. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005), Vancouver, January 2005, pp. 590–601
6. Demaine, E.D., Hajiaghayi, M., Kawarabayashi, K.-I.: Algorithmic graph minor theory: Decomposition, approximation, and coloring. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, Pittsburgh, October 2005, pp. 637–646
7. Demaine, E.D., Hajiaghayi, M., Mohar, B.: Approximation algorithms via contraction decomposition. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, 7–9 January 2007, pp. 278–287
8. Demaine, E.D., Hajiaghayi, M., Nishimura, N., Ragde, P., Thilikos, D.M.: Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. J. Comput. Syst. Sci. **69**(2), 166–195 (2004)
9. DeVos, M., Ding, G., Oporowski, B., Sanders, D.P., Reed, B., Seymour, P., Vertigan, D.: Excluding any graph as a minor allows a low tree-width 2-coloring. J. Comb. Theory Ser. B **91**(1), 25–41 (2004)
10. Eppstein, D.: Diameter and treewidth in minor-closed graph families. Algorithmica **27**(3–4), 275–291 (2000)
11. Frick, M., Grohe, M.: Deciding first-order properties of locally tree-decomposable structures. J. ACM **48**(6), 1184–1206 (2001)
12. Grohe, M.: Local tree-width, excluded minors, and approximation algorithms. Combinatorica **23**(4), 613–632 (2003)
13. Klein, P.N.: A linear-time approximation scheme for TSP for planar weighted graphs. In: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science, 2005, pp. 146–155
14. Klein, P.N.: A subset spanner for planar graphs, with application to subset TSP. In: Proceedings of the 38th ACM Symposium on Theory of Computing, 2006, pp. 749–756

15. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM J. Appl. Math. **36**(2), 177–189 (1979)
16. Lipton, R.J., Tarjan, R.E.: Applications of a planar separator theorem. SIAM J. Comput. **9**(3), 615–627 (1980)

Arbitrage in Frictional Foreign Exchange Market

2003; Cai, Deng

MAO-CHENG CAI[1], XIAOTIE DENG[2]
[1] Institute of Systems Science, Chinese Academy of Sciences, Beijing, China
[2] Department of Computer Science, City University of Hong Kong, Hong Kong, China

Problem Definition

The simultaneous purchase and sale of the same securities, commodities, or foreign exchange in order to profit from a differential in the price. This usually takes place on different exchanges or marketplaces. Also known as a "Riskless profit".

Arbitrage is, arguably, the most fundamental concept in finance. It is a state of the variables of financial instruments such that a riskless profit can be made, which is generally believed not in existence. The economist's argument for its non-existence is that active investment agents will exploit any arbitrage opportunity in a financial market and thus will deplete it as soon as it may arise. Naturally, the speed at which such an arbitrage opportunity can be located and be taken advantage of is important for the profit-seeking investigators, which falls in the realm of analysis of algorithms and computational complexity.

The identification of arbitrage states is, at frictionless foreign exchange market (a theoretical trading environment where all costs and restraints associated with transactions are non-existent), not difficult at all and can be reduced to existence of arbitrage on three currencies (see [11]). In reality, friction does exist. Because of friction, it is possible that there exist arbitrage opportunities in the market but difficult to find it and to exploit it to eliminate it. Experimental results in foreign exchange markets showed that arbitrage does exist in reality. Examination of data from ten markets over a twelve day period by Mavrides [11] revealed that a significant arbitrage opportunity exists. Some opportunities were observed to be persistent for a long time. The problem become worse at forward and futures markets (in which futures contracts in commodities are traded) coupled with covered interest rates, as observed by Abeysekera and Turtle [1], and Clinton [4]. An obvious interpretation is that the arbitrage opportunity was not immediately identified because of information asymmetry in the market. However, that is not the only factor. Both the time necessary to collect the market information (so that an arbitrage opportunity would be identified) and the time people (or computer programs) need to find the arbitrage transactions are important factors for eliminating arbitrage opportunities.

The computational complexity in identifying arbitrage, the level in difficulty measured by arithmetic operations, is different in different models of exchange systems. Therefore, to approximate an ideal exchange market, models with lower complexities should be preferred to those with higher complexities.

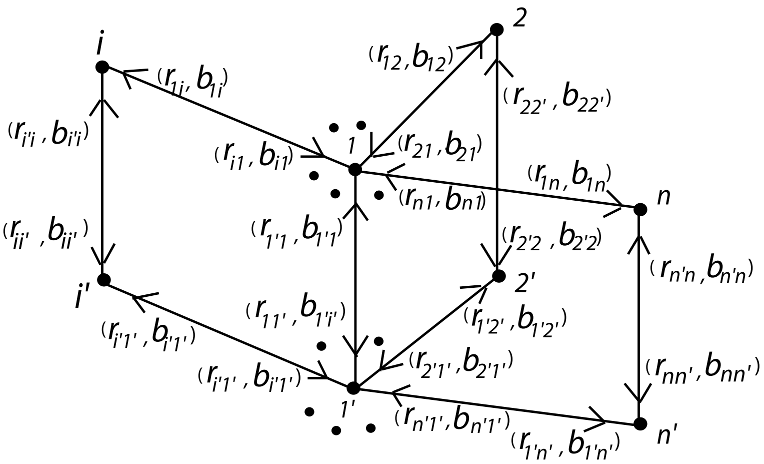
To model an exchange system, consider n foreign currencies: $N = \{1, 2, \dots, n\}$. For each ordered pair (i, j), one may change one unit of currency i to r_{ij} units of currency j. Rate r_{ij} is the *exchange rate* from i to j. In an ideal market, the exchange rate holds for any amount that is exchanged. An *arbitrage opportunity* is a set of exchanges between pairs of currencies such that the net balance for each involved currency is non-negative and there is at least one currency for which the net balance is positive. Under ideal market conditions, there is no arbitrage if and only if there is no arbitrage among any three currencies (see [11]).

Various types of *friction* can be easily modeled in such a system. Bid-offer spread may be expressed in the present mathematical format as $r_{ij} r_{ji} < 1$ for some $i, j \in N$. In addition, usually the traded amount is required to be in multiples of a fixed integer amount, hundreds, thousands or millions. Moreover, different traders may bid or offer at different rates, and each for a limited amount. A more general model to describe these market *imperfections* will include, for pairs $i \neq j \in N$, l_{ij} different rates r^k_{ij} of exchanges from currency i to j up to b^k_{ij} units of currency i, $k = 1, \dots, l_{ij}$, where l_{ij} is the number of different exchange rates from currency i to j.

A currency exchange market can be represented by a digraph $G = (V, E)$ with vertex set V and arc set E such that each vertex $i \in V$ represents currency i and each arc $a^k_{ij} \in E$ represents the currency exchange relation from i to j with rate r^k_{ij} and bound b^k_{ij}. Note that parallel arcs may occur for different exchange rates. Such a digraph is called an exchange digraph. Let $x = (x^k_{ij})$ denote a currency exchange vector.

Problem 1 *The existence of arbitrage in a frictional exchange market can be formulated as follows.*

$$\sum_{j \neq i} \sum_{k=1}^{l_{ji}} \lfloor r^k_{ji} x^k_{ji} \rfloor - \sum_{j \neq i} \sum_{k=1}^{l_{ij}} x^k_{ij} \geq 0, \; i = 1, \dots, n, \; , \quad (1)$$



Arbitrage in Frictional Foreign Exchange Market, Figure 1
Digraph G_1

at least one strict inequality holds

$$0 \le x_{ij}^k \le b_{ij}^k,\ 1 \le k \le l_{ij},\ 1 \le i \ne j \le n\,, \tag{2}$$

$$x_{ij}^k \text{ is integer},\ 1 \le k \le l_{ij},\ 1 \le i \ne j \le n. \tag{3}$$

Note that the first term in the right hand side of (1) is the revenue at currency i by selling other currencies and the second term is the expense at currency i by buying other currencies.

The corresponding optimization problem is

Problem 2 *The maximum arbitrage problem in a frictional foreign exchange market with bid-ask spreads, bound and integrality constraints is the following integer linear programming (P):*

$$\text{maximize} \sum_{i=1}^{n} w_i \sum_{j \ne i} \left(\sum_{k=1}^{l_{ji}} \lfloor r_{ji}^k x_{ji}^k \rfloor - \sum_{k=1}^{l_{ij}} x_{ij}^k \right)$$

subject to

$$\sum_{j \ne i} \left(\sum_{k=1}^{l_{ji}} \lfloor r_{ji}^k x_{ji}^k \rfloor - \sum_{k=1}^{l_{ij}} x_{ij}^k \right) \ge 0\,, \quad i = 1, \ldots, n\,, \tag{4}$$

$$0 \le x_{ij}^k \le b_{ij}^k\,, \quad 1 \le k \le l_{ij}\,, \quad 1 \le i \ne j \le n\,, \tag{5}$$

$$x_{ij}^k \text{ is integer}\,, \quad 1 \le k \le l_{ij}\,, \quad 1 \le i \ne j \le n\,, \tag{6}$$

where $w_i \ge 0$ is a given weight for currency i, $i = 1, 2, \ldots, n$, with at least one $w_i > 0$.

Finally consider another

Problem 3 *In order to eliminate arbitrage, how many transactions and arcs in a exchange digraph have to be used for the currency exchange system?*

Key Results

A decision problem is called nondeterministic polynomial (*NP* for short) if its solution (if one exists) can be guessed and verified in polynomial time; nondeterministic means that no particular rule is followed to make the guess. If a problem is *NP* and all other *NP* problems are polynomial-time reducible to it, the problem is *NP*-complete. And a problem is called *NP*-hard if every other problem in NP is polynomial-time reducible to it.

Theorem 1 *It is NP-complete to determine whether there exists arbitrage in a frictional foreign exchange market with bid-ask spreads, bound and integrality constraints even if all $l_{ij} = 1$.*

Then a further inapproximability result is obtained.

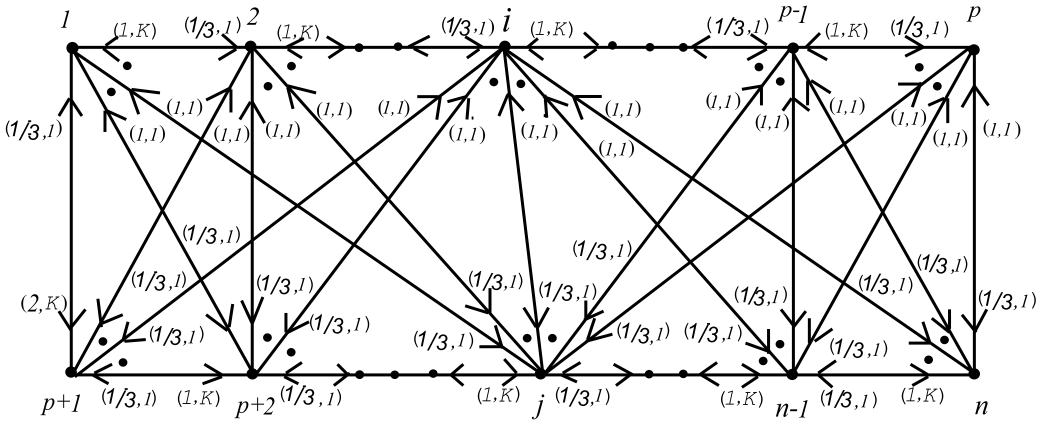
Theorem 2 *There exists fixed $\epsilon > 0$ such that approximating (P) within a factor of n^ϵ is NP-hard even for any of the following two special cases:*

(P_1) all $l_{ij} = 1$ and $w_i = 1$.
(P_2) all $l_{ij} = 1$ and all but one $w_i = 0$.

Now consider two polynomially solvable special cases when the number of currencies is constant or the exchange digraph is star-shaped (a digraph is star-shaped if all arcs have a common vertex).

Theorem 3 *There are polynomial time algorithms for (P) when the number of currencies is constant.*

Theorem 4 *It is polynomially solvable to find the maximum revenue at the center currency of arbitrage in a frictional foreign exchange market with bid-ask spread, bound*



Arbitrage in Frictional Foreign Exchange Market, Figure 2
Digraph G_2

and integrality constraints when the exchange digraph is star-shaped.

However, if the exchange digraph is the coalescence of a star-shaped exchange digraph and its copy, shown by Digraph G_1, then the problem becomes NP-complete.

Theorem 5 *It is NP-complete to decide whether there exists arbitrage in a frictional foreign exchange market with bid-ask spreads, bound and integrality constraints even if its exchange digraph is coalescent.*

Finally an answer to Problem 3 is as follows.

Theorem 6 *There is an exchange digraph of order n such that at least $\lfloor n/2 \rfloor \lceil n/2 \rceil - 1$ transactions and at least $n^2/4 + n - 3$ arcs are in need to bring the system back to non-arbitrage states.*

For instance, consider the currency exchange market corresponding to digraph $G_2 = (V, E)$, where the number of currencies is $n = |V|$, $p = \lfloor n/2 \rfloor$ and $K = n^2$.

Set

$$\begin{aligned} C = &\{a_{ij} \in E \mid 1 \le i \le p, p+1 \le j \le n\} \\ &\cup \{a_{1(p+1)}\} \setminus \{a_{(p+1)1}\} \cup \{a_{i(i-1)} \mid 2 \le i \le p\} \\ &\cup \{a_{i(i+1)} \mid p+1 \le i \le n-1\} . \end{aligned}$$

Then $|C| = \lfloor n/2 \rfloor \lceil n/2 \rceil + n - 2 = |E|/2 > n^2/4 + n - 3$. It follows easily from the rates and bounds that each arc in C has to be used to eliminate arbitrage. And $\lfloor n/2 \rfloor \lfloor n/2 \rceil - 1$ transactions corresponding to $\{a_{ij} \in E \mid 1 \le i \le p, p + 1 \le j \le n\} \setminus \{a_{(p+1)1}\}$ are in need to bring the system back to non-arbitrage states.

Applications

The present results show that different foreign exchange systems exhibit quite different computational complexities. They may shed new light on how monetary system models are adopted and evolved in reality. In addition, it provides with a computational complexity point of view to the understanding of the now fast growing Internet electronic exchange markets.

Open Problems

The dynamic models involving in both spot markets (in which goods are sold for cash and delivered immediately) and futures markets are the most interesting ones. To develop good approximation algorithms for such general models would be important. In addition, it is also important to identify special market models for which polynomial time algorithms are possible even with future markets. Another interesting paradox in this line of study is why friction constraints that make arbitrage difficult are not always eliminated in reality.

Cross References

► General Equilibrium

Recommended Reading

1. Abeysekera, S.P., Turtle H.J.: Long-run relations in exchange markets: a test of covered interest parity. J. Financial Res. **18**(4), 431–447 (1995)
2. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and approximation: combinatorial optimization problems and their approximability properties. Springer, Berlin (1999)

3. Cai, M., Deng, X.: Approximation and computation of arbitrage in frictional foreign exchange market. Electron. Notes Theor. Comput. Sci. **78**, 1–10(2003)
4. Clinton, K.: Transactions costs and covered interest arbitrage: theory and evidence. J. Politcal Econ. **96**(2), 358–370 (1988)
5. Deng, X., Li, Z.F., Wang, S.: Computational complexity of arbitrage in frictional security market. Int. J. Found. Comput. Sci. **13**(5), 681–684 (2002)
6. Deng, X., Papadimitriou, C.: On the complexity of cooperative game solution concepts. Math. Oper. Res. **19**(2), 257–266 (1994)
7. Deng, X., Papadimitriou, C., Safra, S.: On the complexity of price equilibria. J. Comput. System Sci. **67**(2), 311–324 (2003)
8. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide of the theory of *NP*-completeness. Freeman, San Francisco (1979)
9. Jones, C.K.: A network model for foreign exchange arbitrage, hedging and speculation. Int. J. Theor. Appl. Finance **4**(6), 837–852 (2001)
10. Lenstra Jr., H.W.: Integer programming with a fixed number of variables. Math. Oper. Res. **8**(4), 538–548 (1983)
11. Mavrides, M.: Triangular arbitrage in the foreign exchange market – inefficiencies, technology and investment opportunities. Quorum Books, London (1992)
12. Megiddo, N.: Computational complexity and the game theory approach to cost allocation for a tree. Math. Oper. Res. **3**, 189–196 (1978)
13. Mundell, R.A.: Currency areas, exchange rate systems, and international monetary reform, paper delivered at Universidad del CEMA, Buenos Aires, Argentina. http://www.robertmundell.net/pdf/Currency (2000). Accessed 17 Apr 2000
14. Mundell, R.A.: Gold Would Serve into the 21st Century. Wall Street Journal, 30 September 1981, pp. 33
15. Zhang, S., Xu, C., Deng, X.: Dynamic arbitrage-free asset pricing with proportional transaction costs. Math. Finance **12**(1), 89–97 (2002)

Arithmetic Coding for Data Compression

1994; Howard, Vitter

Paul G. Howard[1], Jeffrey Scott Vitter[2]

[1] Microway, Inc., Plymouth, MA, USA

[2] Department of Computer Science, Purdue University, West Lafayette, IN, USA

Keywords and Synonyms

Entropy coding; Statistical data compression

Problem Definition

Often it is desirable to encode a sequence of data efficiently to minimize the number of bits required to transmit or store the sequence. The sequence may be a file or message consisting of *symbols* (or *letters* or *characters*) taken from a fixed input alphabet, but more generally the sequence can be thought of as consisting of *events*, each taken from its own input set. Statistical data compression is concerned with encoding the data in a way that makes use of probability estimates of the events. Lossless compression has the property that the input sequence can be reconstructed exactly from the encoded sequence. Arithmetic coding is a nearly-optimal statistical coding technique that can produce a lossless encoding.

Problem (Statistical data compression)

INPUT: *A sequence of m events $a_1, a_2, \dots, a_m$. The ith event a_i is taken from a set of n distinct possible events $e_{i,1}, e_{i,2}, \dots, e_{i,n}$, with an accurate assessment of the probability distribution P_i of the events. The distributions P_i need not be the same for each event a_i.*

OUTPUT: *A succinct encoding of the events that can be decoded to recover exactly the original sequence of events.*

The goal is to achieve optimal or near-optimal encoding length. Shannon [10] proved that the smallest possible expected number of bits needed to encode the ith event is the *entropy* of P_i, denoted by

$$H(P_i) = \sum_{k=1}^{n} -p_{i,k} \log_2 p_{i,k}$$

where $p_{i,k}$ is the probability that e_k occurs as the ith event. An optimal code outputs $-\log_2 p$ bits to encode an event whose probability of occurrence is p.

The well-known Huffman codes [6] are optimal only among *prefix* (or *instantaneous*) codes, that is, those in which the encoding of one event can be decoded before encoding has begun for the next event. Hu–Tucker codes are prefix codes similar to Huffman codes, and are derived using a similar algorithm, with the added constraint that coded messages preserve the ordering of original messages.

When an instantaneous code is not needed, as is often the case, arithmetic coding provides a number of benefits, primarily by relaxing the constraint that the code lengths must be integers: 1) The code length is optimal ($-\log_2 p$ bits for an event with probability p), even when probabilities are not integer powers of $\frac{1}{2}$. 2) There is no loss of coding efficiency even for events with probability close to 1. 3) It is trivial to handle probability distributions that change from event to event. 4) The input message to output message ordering correspondence of Hu–Tucker coding can be obtained with minimal extra effort.

As an example, consider a 5-symbol input alphabet. Symbol probabilities, codes, and code lengths are given in Table 1.

The average code length is 2.13 bits per input symbol for the Huffman code, 2.22 bits per symbol for the Hu–

Arithmetic Coding for Data Compression, Table 1
Comparison of codes for Huffman coding, Hu-Tucker coding, and arithmetic coding for a sample 5-symbol alphabet

Symbol	Prob.		Huffman		Hu–Tucker		Arithmetic
e_k	p_k	$-\log_2 p_k$	Code	Length	Code	Length	Length
a	0.04	4.644	**1111**	4	**000**	3	4.644
b	0.18	2.474	**110**	3	**001**	3	2.474
c	0.43	1.218	**0**	1	**01**	2	1.218
d	0.15	2.737	**1110**	4	**10**	2	2.737
e	0.20	2.322	**10**	2	**11**	2	2.322

Tucker code, and 2.03 bits per symbol for arithmetic coding.

Key Results

In theory, arithmetic codes assign one "codeword" to each possible input sequence. The codewords consist of half-open subintervals of the half-open unit interval $[0, 1)$, and are expressed by specifying enough bits to distinguish the subinterval corresponding to the actual sequence from all other possible subintervals. Shorter codes correspond to larger subintervals and thus more probable input sequences. In practice, the subinterval is refined incrementally using the probabilities of the individual events, with bits being output as soon as they are known. Arithmetic codes almost always give better compression than prefix codes, but they lack the direct correspondence between the events in the input sequence and bits or groups of bits in the coded output file.

The algorithm for encoding a file using arithmetic coding works conceptually as follows:

1. The "current interval" $[L, H)$ is initialized to $[0, 1)$.
2. For each event in the file, two steps are performed.
   (a) Subdivide the current interval into subintervals, one for each possible event. The size of a event's subinterval is proportional to the estimated probability that the event will be the next event in the file, according to the model of the input.
   (b) Select the subinterval corresponding to the event that actually occurs next and make it the new current interval.
3. Output enough bits to distinguish the final current interval from all other possible final intervals.

The length of the final subinterval is clearly equal to the product of the probabilities of the individual events, which is the probability p of the particular overall sequence of events. It can be shown that $\lfloor -\log_2 p \rfloor + 2$ bits are enough to distinguish the file from all other possible files.

For finite-length files, it is necessary to indicate the end of the file. In arithmetic coding this can be done easily by introducing a special low-probability event that can be be injected into the input stream at any point. This adds only $O(\log m)$ bits to the encoded length of an m-symbol file.

In step 2, one needs to compute only the subinterval corresponding to the event a_i that actually occurs. To do this, it is convenient to use two "cumulative" probabilities: the cumulative probability $P_C = \sum_{k=1}^{i-1} p_k$ and the next-cumulative probability $P_N = P_C + p_i = \sum_{k=1}^{i} p_k$. The new subinterval is $[L + P_C(H - L), L + P_N(H - L))$. The need to maintain and supply cumulative probabilities requires the model to have a sophisticated data structure, such as that of Moffat [7], especially when many more than two events are possible.

Modeling

The goal of modeling for statistical data compression is to provide probability information to the coder. The modeling process consists of structural and probability estimation components; each may be adaptive (starting from a neutral model, gradually build up the structure and probabilities based on the events encountered), semi-adaptive (specify an initial model that describes the events to be encountered in the data, then modify the model during coding so that it describes only the events yet to be coded), or static (specify an initial model, and use it without modification during coding).

In addition there are two strategies for probability estimation. The first is to estimate each event's probability individually based on its frequency within the input sequence. The second is to estimate the probabilities collectively, assuming a probability distribution of a particular form and estimating the parameters of the distribution, either directly or indirectly. For direct estimation, the data can yield an estimate of the parameter (the variance, for instance). For indirect estimation [5], one can start with a small number of possible distributions and compute the code length that would be obtained with each; the one with the smallest code length is selected. This method is very general and can be used even for distributions from different families, without common parameters.

Arithmetic coding is often applied to text compression. The events are the symbols in the text file, and the model consists of the probabilities of the symbols considered in some context. The simplest model uses the overall frequencies of the symbols in the file as the probabilities; this is a zero-order Markov model, and its entropy is denoted H_0. The probabilities can be estimated adaptively starting with counts of 1 for all symbols and incrementing after each symbol is coded, or the symbol counts can be coded before coding the file itself and either modified during coding (a decrementing semi-adaptive code) or left unchanged (a static code). In all cases, the code length is independent of the order of the symbols in the file.

Theorem 1 *For all input files, the code length L_A of an adaptive code with initial 1-weights is the same as the code length L_{SD} of the semi-adaptive decrementing code plus the code length L_M of the input model encoded assuming that all symbol distributions are equally likely. This code length is less than $L_S = mH_0 + L_M$, the code length of a static code with the same input model. In other words, $L_A = L_{SD} + L_M < mH_0 + L_M = L_S$.*

It is possible to obtain considerably better text compression by using higher order Markov models. Cleary and Witten [2] were the first to do this with their PPM method. PPM requires adaptive modeling and coding of probabilities close to 1, and makes heavy use of arithmetic coding.

Implementation Issues

Incremental Output. The basic implementation of arithmetic coding described above has two major difficulties: the shrinking current interval requires the use of high precision arithmetic, and no output is produced until the entire file has been read. The most straightforward solution to both of these problems is to output each leading bit as soon as it is known, and then to double the length of the current interval so that it reflects only the unknown part of the final interval. Witten, Neal, and Cleary [11] add a clever mechanism for preventing the current interval from shrinking too much when the endpoints are close to $\frac{1}{2}$ but straddle $\frac{1}{2}$. In that case one does not yet know the next output bit, but whatever it is, the *following* bit will have the opposite value; one can merely keep track of that fact, and expand the current interval symmetrically about $\frac{1}{2}$. This follow-on procedure may be repeated any number of times, so the current interval size is always strictly longer than $\frac{1}{4}$.

Before [11] other mechanisms for incremental transmission and fixed precision arithmetic were developed through the years by a number of researchers beginning with Pasco [8]. The bit-stuffing idea of Langdon and others at IBM [9] that limits the propagation of carries in the additions serves a function similar to that of the follow-on procedure described above.

Use of Integer Arithmetic. In practice, the arithmetic can be done by storing the endpoints of the current interval as sufficiently large integers rather than in floating point or exact rational numbers. Instead of starting with the real interval $[0, 1)$, start with the integer interval $[0, N)$, N invariably being a power of 2. The subdivision process involves selecting non-overlapping integer intervals (of length at least 1) with lengths approximately proportional to the counts.

Limited-Precision Arithmetic Coding. Arithmetic coding as it is usually implemented is slow because of the multiplications (and in some implementations, divisions) required in subdividing the current interval according to the probability information. Since small errors in probability estimates cause very small increases in code length, introducing approximations into the arithmetic coding process in a controlled way can improve coding speed without significantly degrading compression performance. In the Q-Coder work at IBM [9], the time-consuming multiplications are replaced by additions and shifts, and low-order bits are ignored.

Howard and Vitter [4] describe a different approach to approximate arithmetic coding. The fractional bits characteristic of arithmetic coding are stored as state information in the coder. The idea, called *quasi-arithmetic coding*, is to reduce the number of possible states and replace arithmetic operations by table lookups; the lookup tables can be precomputed.

The number of possible states (after applying the interval expansion procedure) of an arithmetic coder using the integer interval $[0, N)$ is $3N^2/16$. The obvious way to reduce the number of states in order to make lookup tables practicable is to reduce N. Binary quasi-arithmetic coding causes an insignificant increase in the code length compared with pure arithmetic coding.

Theorem 2 *In a quasi-arithmetic coder based on full interval $[0, N)$, using correct probability estimates, and excluding very large and very small probabilities, the number of bits per input event by which the average code length obtained by the quasi-arithmetic coder exceeds that of an ex-*

act arithmetic coder is at most

$$\frac{4}{\ln 2}\left(\log_2 \frac{2}{e \ln 2}\right)\frac{1}{N} + O\left(\frac{1}{N^2}\right) \approx \frac{0.497}{N} + O\left(\frac{1}{N^2}\right) ,$$

and the fraction by which the average code length obtained by the quasi-arithmetic coder exceeds that of an exact arithmetic coder is at most

$$\left(\log_2 \frac{2}{e \ln 2}\right)\frac{1}{\log_2 N} + O\left(\frac{1}{(\log N)^2}\right) \approx \frac{0.0861}{\log_2 N} + O\left(\frac{1}{(\log N)^2}\right) .$$

General-purpose algorithms for parallel encoding and decoding using both Huffman and quasi-arithmetic coding are given in [3].

Applications

Arithmetic coding can be used in most applications of data compression. Its main usefulness is in obtaining maximum compression in conjunction with an adaptive model, or when the probability of one event is close to 1. Arithmetic coding has been used heavily in text compression. It has also been used in image compression in the JPEG international standards for image compression and is an essential part of the JBIG international standards for bilevel image compression. Many fast implementations of arithmetic coding, especially for a two-symbol alphabet, are covered by patents; considerable effort has been expended in adjusting the basic algorithm to avoid infringing those patents.

Open Problems

The technical problems with arithmetic coding itself have been completely solved. The remaining unresolved issues are concerned with modeling: decomposing an input data set into a sequence of events, the set of events possible at each point in the data set being described by a probability distribution suitable for input into the coder. The modeling issues are entirely application-specific.

Experimental Results

Some experimental results for the Calgary and Canterbury corpora are summarized in a report by Arnold and Bell [1].

Data Sets

Among the most widely used data sets suitable for research in arithmetic coding are: the Calgary Corpus: (ftp://ftp.cpsc.ucalgary.ca/pub/projects), the Canterbury Corpus (corpus.canterbury.ac.nz), and the Pizza&Chili Corpus (pizzachili.dcc.uchile.cl).

URL to Code

A number of implementations of arithmetic coding are available on the Compression Links Info page, www.compression-links.info/ArithmeticCoding. The code at the ucalgary.ca FTP site, based on [11], is especially useful for understanding arithmetic coding.

Cross References

► Boosting Textual Compression
► Burrows–Wheeler Transform

Recommended Reading

1. Arnold, R., Bell, T.: A corpus for the evaluation of lossless compression algorithms. In: Proceedings of the IEEE Data Compression Conference, Snowbird, Utah, March 1997, pp. 201–210
2. Cleary, J.G., Witten, I.H.: Data compression using adaptive coding and partial string matching. IEEE Transactions on Communications, COM–32, pp. 396–402 (1984)
3. Howard, P.G., Vitter, J.S.: Parallel lossless image compression using Huffman and arithmetic coding. In: Proceedings of the IEEE Data Compression Conference, Snowbird, Utah, March 1992, pp. 299–308
4. Howard, P.G., Vitter, J.S.: Practical implementations of arithmetic coding. In: Storer, J.A. (ed.) Images and Text Compression. Kluwer Academic Publishers, Norwell, Massachusetts (1992)
5. Howard, P.G., Vitter, J.S.: Fast and efficient lossless image compression. In: Proceedings of the IEEE Data Compression Conference, Snowbird, Utah, March 1993, pp. 351–360
6. Huffman, D.A.: A method for the construction of minimum redundancy codes. Proceedings of the Institute of Radio Engineers, 40, pp. 1098–1101 (1952)
7. Moffat, A.: An improved data structure for cumulative probability tables. Softw. Prac. Exp. **29**, 647–659 (1999)
8. Pasco, R.: Source Coding Algorithms for Fast Data Compression, Ph. D. thesis, Stanford University (1976)
9. Pennebaker, W.B., Mitchell, J.L., Langdon, G.G., Arps, R.B.: An overview of the basic principles of the Q-coder adaptive binary arithmetic coder. IBM J. Res. Develop. **32**, 717–726 (1988)
10. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**, 398–403 (1948)
11. Witten, I.H., Neal, R.M., Cleary, J.G.: Arithmetic coding for data compression. Commun. ACM **30**, 520–540 (1987)

Assignment Problem

1955; Kuhn
1957; Munkres

SAMIR KHULLER
Department of Computer Science,
University of Maryland, College Park, MD, USA

Keywords and Synonyms

Weighted bipartite matching

Problem Definition

Assume that a complete bipartite graph, $G(X, Y, X \times Y)$, with weights $w(x, y)$ assigned to every edge (x, y) is given. A matching M is a subset of edges so that no two edges in M have a common vertex. A perfect matching is one in which all the nodes are matched. Assume that $|X| = |Y| = n$. The **weighted matching problem** is to find a matching with the greatest total weight, where $w(M) = \sum_{e \in M} w(e)$. Since G is a complete bipartite graph, it has a perfect matching. An algorithm that solves the weighted matching problem is due to Kuhn [4] and Munkres [6]. Assume that all edge weights are nonnegative.

Key Results

Define a *feasible vertex labeling* ℓ as a mapping from the set of vertices in G to the reals, where

$$\ell(x) + \ell(y) \geq w(x, y) .$$

Call $\ell(x)$ the label of vertex x. It is easy to compute a feasible vertex labeling as follows:

$$\forall y \in Y \quad \ell(y) = 0$$

and

$$\forall x \in X \quad \ell(x) = \max_{y \in Y} w(x, y) .$$

Define the **equality subgraph**, G_ℓ, to be the spanning subgraph of G, which includes all vertices of G but only those edges (x, y) that have weights such that

$$w(x, y) = \ell(x) + \ell(y) .$$

The connection between equality subgraphs and maximum-weighted matchings is provided by the following theorem.

Theorem 1 *If the equality subgraph, G_ℓ, has a perfect matching, M^*, then M^* is a maximum-weighted matching in G.*

In fact, note that the sum of the labels is an upper bound on the weight of the maximum-weighted perfect matching. The algorithm eventually finds a matching and a feasible labeling such that the weight of the matching is equal to the sum of all the labels.

High-Level Description

The above theorem is the basis of an algorithm for finding a maximum-weighted matching in a complete bipartite graph. Starting with a feasible labeling, compute the equality subgraph and then find a maximum matching in this subgraph (here one can ignore weights on edges). If the matching found is perfect, the process is done. If it is not perfect, more edges are added to the equality subgraph by revising the vertex labels. After adding edges to the equality subgraph, either the size of the matching goes up (an augmenting path is found) or the Hungarian tree continues to grow.[1] In the former case, the phase terminates and a new phase starts (since the matching size has gone up). In the latter case, the Hungarian tree, grows by adding new nodes to it, and clearly this cannot happen more than n times.

Let S be the set of free nodes in X. Grow Hungarian trees from each node in S. Let T be the nodes in Y encountered in the search for an augmenting path from nodes in S. Add all nodes from X that are encountered in the search to S.

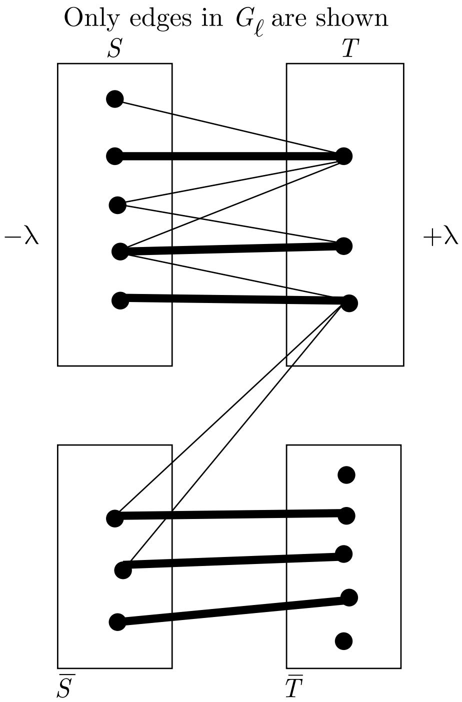
Note the following about this algorithm:

$$\overline{S} = X \setminus S .$$
$$\overline{T} = Y \setminus T .$$
$$|S| > |T| .$$

There are no edges from S to $\overline{T}$ since this would imply that one did not grow the Hungarian trees completely. As the Hungarian trees in are grown in G_ℓ, alternate nodes in the search are placed into S and T. To revise the labels, take the labels in S and start decreasing them uniformly (say, by λ), and at the same time increase the labels in T by λ. This ensures that the edges from S to T do not leave the equality subgraph (Fig. 1).

As the labels in S are decreased, edges (in G) from S to $\overline{T}$ will potentially enter the equality subgraph, G_ℓ. As we increase λ, at some point in time, an edge enters the equality subgraph. This is when one stops and updates the Hungarian tree. If the node from $\overline{T}$ added to T is matched to a node in $\overline{S}$, both these nodes are moved to S and T, which yields a larger Hungarian tree. If the node from $\overline{T}$ is free, an augmenting path is found and the phase is complete. One phase consists of those steps taken between increases in the size of the matching. There are at most n phases, where n is the number of vertices in G (since in each phase

[1] This is the structure of explored edges when one starts BFS simultaneously from all free nodes in S. When one reaches a matched node in T, one only explores the matched edge; however, all edges incident to nodes in S are explored.



Assignment Problem, Figure 1
Sets *S* and *T* as maintained by the algorithm

the size of the matching increases by 1). Within each phase the size of the Hungarian tree is increased at most n times. It is clear that in $O(n^2)$ time one can figure out which edge from S to $\overline{T}$ is the first to enter the equality subgraph (one simply scans all the edges). This yields an $O(n^4)$ bound on the total running time. How to implement it in $O(n^3)$ time is now shown.

More Efficient Implementation

Define the slack of an edge as follows:

$$\mathit{slack}(x,y) = \ell(x) + \ell(y) - w(x,y) \,.$$

Then

$$\lambda = \min_{x \in S, y \in \overline{T}} \mathit{slack}(x,y) \,.$$

Naively, the calculation of λ requires $O(n^2)$ time. For every vertex $y \in \overline{T}$, keep track of the edge with the smallest slack, i. e.,

$$\mathit{slack}[y] = \min_{x \in S} \mathit{slack}(x,y) \,.$$

The computation of $\mathit{slack}[y]$ (for all $y \in \overline{T}$) requires $O(n^2)$ time at the start of a phase. As the phase progresses, it is easy to update all the *slack* values in $O(n)$ time since all of them change by the same amount (the labels of the vertices in S are going down uniformly). Whenever a node u is moved from $\overline{S}$ to S one must recompute the slacks of the nodes in $\overline{T}$, requiring $O(n)$ time. But a node can be moved from $\overline{S}$ to S at most n times.

Thus each phase can be implemented in $O(n^2)$ time. Since there are n phases, this gives a running time of $O(n^3)$. For sparse graphs, there is a way to implement the algorithm in $O(n(m + n \log n))$ time using min cost flows [1], where m is the number of edges.

Applications

There are numerous applications of biparitite matching, for example, scheduling unit-length jobs with integer release times and deadlines, even with time-dependent penalties.

Open Problems

Obtaining a linear, or close to linear, time algorithm.

Recommended Reading

Several books on combinatorial optimization describe algorithms for weighted bipartite matching (see [2,5]). See also Gabow's paper [3].

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows: Theory, Algorithms and Applications. Prentice Hall, Englewood Cliffs (1993)
2. Cook, W., Cunningham, W., Pulleyblank, W., Schrijver, A.: Combinatorial Optimization. Wiley, New York (1998)
3. Gabow, H.: Data structures for weighted matching and nearest common ancestors with linking. In: Symp. on Discrete Algorithms, 1990, pp. 434–443
4. Kuhn, H.: The Hungarian method for the assignment problem. Naval Res. Logist. Quart. **2**, 83–97 (1955)
5. Lawler, E.: Combinatorial Optimization: Networks and Matroids. Holt, Rinehart and Winston (1976)
6. Munkres, J.: Algorithms for the assignment and transportation problems. J. Soc. Ind. Appl. Math. **5**, 32–38 (1957)

Asynchronous Consensus Impossibility

1985; Fischer, Lynch, Paterson

MAURICE HERLIHY
Department of Computer Science, Brown University,
Providence, RI, USA

Keywords and Synonyms

Wait-free consensus; Agreement

Problem Definition

Consider a distributed system consisting of a set of *processes* that communicate by sending and receiving messages. The network is a multiset of messages, where each message is addressed to some process. A process is a state machine that can take three kinds of *steps*.

- In a *send* step, a process places a message in the network.
- In a *receive* step, a process A either reads and removes from the network a message addressed to A, or it reads a distinguished *null* value, leaving the network unchanged. If a message addressed to A is placed in the network, and if A subsequently performs an infinite number of receive steps, then A will eventually receive that message.
- In a *computation* state, a process changes state without communicating with any other process.

Processes are *asynchronous*: there is no bound on their relative speeds. Processes can *crash*: they can simply halt and take no more steps. This article considers executions in which at most one process crashes.

In the *consensus* problem, each process starts with a private *input* value, communicates with the others, and then halts with a *decision* value. These values must satisfy the following properties:

- *Agreement:* all processes' decision values must agree.
- *Validity:* every decision value must be some process' input.
- *Termination:* every non-fault process must decide in a finite number of steps.

Fischer, Lynch, and Paterson showed that there is no protocol that solves consensus in any asynchronous message-passing system where even a single process can fail. This result is one of the most influential results in Distributed Computing, laying the foundations for a number of subsequent research efforts.

Terminology

Without loss of generality, one can restrict attention to *binary* consensus, where the inputs are 0 or 1. A *protocol state* consists of the states of the processes and the multiset of messages in transit in the network. An *initial state* is a protocol state before any process has moved, and a *final state* is a protocol state after all processes have finished. The *decision value* of any final state is the value decided by all processes in that state.

Any terminating protocol's set of possible states forms a tree, where each node represents a possible protocol state, and each edge represents a possible step by some process. Because the protocol must terminate, the tree is finite. Each leaf node represents a final protocol state with decision value either 0 or 1.

A *bivalent* protocol state is one in which the eventual decision value is not yet fixed. From any bivalent state, there is an execution in which the eventual decision value is 0, and another in which it is 1. A *univalent* protocol state is one in which the outcome is fixed. Every execution starting from a univalent state decides the same value. A *1-valent* protocol state is univalent with eventual decision value 1, and similarly for a *0-valent* state.

A protocol state is *critical* if

- It is bivalent, and
- If any process takes a step, the protocol state becomes univalent.

Key Results

Lemma 1 *Every consensus protocol has a bivalent initial state.*

Proof Assume, by way of contradiction, that there exists a consensus protocol for $(n+1)$ threads $A_0, \cdots, A_n$ in which every initial state is univalent. Let s_i be the initial state where processes $A_i, \cdots, A_n$ have input 0 and $A_0, \ldots, A_{i-1}$ have input 1. Clearly, s_0 is 0-valent: all processes have input 0, so all must decide 0 by the validity condition. If s_i is 0-valent, so is s_{i+1}. These states differ only in the input to process A_i : 0 in s_i, and 1 in s_{i+1}. Any execution starting from s_i in which A_i halts before taking any steps is indistinguishable from an execution starting from s_{i+1} in which A_i halts before taking any steps. Since processes must decide 0 in the first execution, they must decide 1 in the second. Since there is one execution starting from s_{i+1} that decides 0, and since s_{i+1} is univalent by hypothesis, s_{i+1} is 0-valent. It follows that the state s_{n+1}, in which all processes start with input 1, is 0-valent, a contradiction. □

Lemma 2 *Every consensus protocol has a critical state.*

Proof by contradiction. By Lemma 1, the protocol has a bivalent initial state. Start the protocol in this state. Repeatedly choose a process whose next step leaves the protocol in a bivalent state, and let that process take a step. Either the protocol runs forever, violating the termination condition, or the protocol eventually enters a critical state. □

Theorem 3 *There is no consensus protocol for an asynchronous message-passing system where a single process can crash.*

Proof Assume by way of contradiction that such a protocol exists. Run the protocol until it reaches a critical state

s. There must be two processes A and B such that A's next step carries the protocol to a 0-valent state, and B's next step carries the protocol to a 1-valent state.

Starting from s, let s_A be the state reached if A takes the first step, s_B if B takes the first step, s_{AB} if A takes a step followed by B, and so on. States s_A and s_{AB} are 0-valent, while s_B and s_{BA} are 1-valent. The rest is a case analysis.

Of all the possible pairs of steps A and B could be about to execute, most of them *commute*: states s_{AB} and s_{BA} are identical, which is a contradiction because they have different valences.

The only pair of steps that do not commute occurs when A is about to send a message to B (or vice versa). Let s_{AB} be the state resulting if A sends a message to B and B then receives it, and let s_{BA} be the state resulting if B receives a different message (or *null*) and then A sends its message to B. Note that every process other than B has the same local state in s_{AB} and s_{BA}. Consider an execution starting from s_{AB} in which every process other than B takes steps in round-robin order. Because s_{AB} is 0-valent, they will eventually decide 0. Next, consider an execution starting from s_{BA} in which every process other than B takes steps in round-robin order. Because s_{BA} is 1-valent, they will eventually decide 1. But all processes other than B have the same local states at the end of each execution, so they cannot decide different values, a contradiction. □

In the proof of this theorem, and in the proofs of the preceding lemmas, we construct scenarios where at most a single process is delayed. As a result, this impossibility result holds for any system where a single process can fail undetectably.

Applications

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

Open Problems

There are many open problems concerning the solvability of consensus in other models, or with restrictions on inputs.

Related Work

The original paper by Fischer, Lynch, and Paterson [8] is still a model of clarity.

Many researchers have examined alternative models of computation in which consensus can be solved. Dolev, Dwork, and Stockmeyer [5] examine a variety of alternative message-passing models, identifying the precise assumptions needed to make consensus possible. Dwork, Lynch, and Stockmeyer [6] derive upper and lower bounds for a semi-synchronous model where there is an upper and lower bound on message delivery time. Ben-Or [1] showed that introducing randomization makes consensus possible in an asynchronous message-passing system. Chandra and Toueg [3] showed that consensus becomes possible if in the presence of an oracle that can (unreliably) detect when a process has crashed. Each of the papers cited here has inspired many follow-up papers. A good place to start is the excellent survey by Fich and Ruppert [7].

A protocol is *wait-free* if it tolerates failures by all but one of the participants. A concurrent object implementation is *linearizable* if each method call seems to take effect instantaneously at some point between the method's invocation and response. Herlihy [9] showed that shared-memory objects can each be assigned a *consensus number*, which is the maximum number of processes for which there exists a wait-free consensus protocol using a combination of read-write memory and the objects in question. Consensus numbers induce an infinite hierarchy on objects, where (simplifying somewhat) higher objects are more powerful than lower objects. In a system of n or more concurrent processes, it is impossible to construct a lock-free implementation of an object with consensus number n from an object with a lower consensus number. On the other hand, any object with consensus number n is *universal* in a system of n or fewer processes: it can be used to construct a wait-free linearizable implementation of any object.

In 1990, Chaudhuri [4] introduced the *k-set agreement* problem (sometimes called *k-set consensus*, which generalizes consensus by allowing k or fewer distinct decision values to be chosen. In particular, 1-set agreement is consensus. The question whether k-set agreement can be solved in asynchronous message-passing models was open for several years, until three independent groups [2,10,11] showed that no protocol exists.

Cross References

► Linearizability
► Topology Approach in Distributed Computing

Recommended Reading

1. Ben-Or, M.: Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In: PODC '83: Proceedings of the second annual ACM symposium on Principles of distributed computing, pp. 27–30. ACM Press, New York (1983)

2. Borowsky, E., Gafni, E.: Generalized FLP impossibility result for *t*-resilient asynchronous computations. In: Proceedings of the 1993 ACM Symposium on Theory of Computing, May 1993. pp. 206–215
3. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43**(2), 225–267 (1996)
4. Chaudhuri, S.: Agreement is harder than consensus: Set consensus problems in totally asynchronous systems. In: Proceedings Of The Ninth Annual ACM Symposium On Principles of Distributed Computing, August 1990. pp. 311–234
5. Chandhuri, S.: More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems. Inf. Comput. **105**(1), 132–158, July 1993
6. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
7. Fich, F., Ruppert, E.: Hundreds of impossibility results for distributed computing. Distrib. Comput. **16**(2–3), 121–163 (2003)
8. Fischer, M., Lynch, N., Paterson, M.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)
9. Herlihy, M.: Wait-free synchronization. ACM Trans. Program. Lang. Syst. (TOPLAS) **13**(1), 124–149 (1991)
10. Herlihy, M., Shavit, N.: The topological structure of asynchronous computability. J. ACM **46**(6), 858–923 (1999)
11. Saks, M.E., Zaharoglou, F.: Wait-free k-set agreement is impossible: The topology of public knowledge. SIAM J. Comput. **29**(5), 1449–1483 (2000)

Atomic Broadcast

1995; Cristian, Aghili, Strong, Dolev

XAVIER DÉFAGO
School of Information Science, Japan Advanced Institute
of Science and Technology (JAIST),
Ishikawa, Japan

Keywords and Synonyms

Atomic multicast; Total order broadcast; Total order multicast

Problem Definition

The problem is concerned with allowing a set of processes to concurrently broadcast messages while ensuring that all destinations consistently deliver them in the exact same sequence, in spite of the possible presence of a number of faulty processes.

The work of Cristian, Aghili, Strong, and Dolev [7] considers the problem of atomic broadcast in a system with approximately synchronized clocks and bounded transmission and processing delays. They present successive extensions of an algorithm to tolerate a bounded number of omission, timing, or Byzantine failures, respectively.

Related Work

The work presented in this entry originally appeared as a widely distributed conference contribution [6], over a decade before being published in a journal [7], at which time the work was well-known in the research community. Since there was no significant change in the algorithms, the historical context considered here is hence with respect to the earlier version.

Lamport [11] proposed one of the first published algorithms to solve the problem of ordering broadcast messages in a distributed systems. That algorithm, presented as the core of a mutual exclusion algorithm, operates in a fully asynchronous system (i. e., a system in which there are no bounds on processor speed or communication delays), but does not tolerate failures. Although the algorithms presented here rely on physical clocks rather than Lamport's logical clocks, the principle used for ordering messages is essentially the same: message carry a timestamp of their sending time; messages are delivered in increasing order of the timestamp, using the sending processor name for messages with equal timestamps.

At roughly the same period as the initial publication of the work of Cristian et al. [6], Chang and Maxemchuck [3] proposed an atomic broadcast protocol based on a token passing protocol, and tolerant to crash failures of processors. Also, Carr [1] proposed the Tandem global update protocol, tolerant to crash failures of processors.

Cristian [5] later proposed an extension to the omission-tolerant algorithm presented here, under the assumption that the communication system consists of $f + 1$ independent broadcast channels (where f is the maximal number of faulty processors). Compared with the more general protocol presented here, its extension generates considerably fewer messages.

Since the work of Cristian, Aghili, Strong, and Dolev [7], much has been published on the problem of atomic broadcast (and its numerous variants). For further reading, Défago, Schiper, and Urbán [8] surveyed more than sixty different algorithms to solve the problem, classifying them into five different classes and twelve variants. That survey also reviews many alternative definitions and references about two hundred articles related to this subject. This is still a very active research area, with many new results being published each year.

Hadzilacos and Toueg [10] provide a systematic classification of specifications for variants of atomic broadcast as well as other broadcast problems, such as reliable broadcast, FIFO broadcast, or causal broadcast.

Chandra and Toueg [2] proved the equivalence between atomic broadcast and the *consensus* problem. Thus, any application solved by a consensus can also be solved by atomic broadcast and vice-versa. Similarly, impossibility results apply equally to both problems. For instance, it is well-known that consensus, thus atomic broadcast, cannot be solved deterministically in an asynchronous system with the presence of a faulty process [9].

Notations and Assumptions

The system G consists of n distributed processors and m point-to-point communication links. A link does not necessarily exists between every pair of processors, but it is assumed that the communication network remains connected even in the face of faults (whether processors or links). All processors have distinct names and there exists a total order on them (e. g., lexicographic order).

A component (link or processor) is said to be *correct* if its behavior is consistent with its specification, and *faulty* otherwise. The paper considers three classes of component failures, namely, omission, timing, and Byzantine failures.

- An *omission* failure occurs when the faulty component fails to provide the specified output (e. g., loss of a message).
- A *timing* failure occurs when the faulty component omits a specified output, or provides it either too early or too late.
- A *Byzantine* failure [12] occurs when the component does not behave according to its specification, for instance, by providing output different from the one specified. In particular, the paper considers authentication-detectable Byzantine failures, that is, ones that are detectable using a message authentication protocol, such as error correction codes or digital signatures.

Each processor p has access to a local clock C_p with the properties that (1) two separate clock readings yield different values, and (2) clocks are ε-synchronized, meaning that, at any real time t, the deviation in readings of the clocks of any two processors p and q is at most ε.

In addition, transmission and processing delays, as measured on the clock of a correct processor, are bounded by a known constant δ. This bound accounts not only for delays in transmission and processing, but also for delays due to scheduling, overload, clock drift or adjustments. This is called a synchronous system model.

The diffusion time $d\delta$ is the time necessary to propagate information to all correct processes, in a surviving network of diameter d with the presence of a most π processor failures and λ link failures.

Problem Definition

The problem of atomic broadcast is defined in a synchronous system model as a broadcast primitive which satisfies the following three properties: atomicity, order, and termination.

Problem 1 (Atomic broadcast)
Input: *A stream of messages broadcast by n concurrent processors, some of which may be faulty.*
Output: *The messages delivered in sequence, with the following properties:*

1. *Atomicity: if any correct processor delivers an update at time U on its clock, then that update was initiated by some processor and is delivered by each correct processor at time U on its clock.*
2. *Order: all updates delivered by correct processors are delivered in the same order by each correct processor.*
3. *Termination: every update whose broadcast is initiated by a correct processor at time T on its clock is delivered at all correct processors at time $T + \Delta$ on their clock.*

Nowadays, problem definitions for atomic broadcast that do not explicitly refer to physical time are often preferred. Many variants of time-free definitions are reviewed by Hadzilacos and Toueg [10] and Défago et al. [8]. One such alternate definition is presented below, with the terminology adapted to the context of this entry.

Problem 2 (Total order broadcast)
Input: *A stream of messages broadcast by n concurrent processors, some of which may be faulty.*
Output: *The messages delivered in sequence, with the following properties:*

1. *Validity: if a correct processor broadcasts a message m, then it eventually delivers m.*
2. *Uniform agreement: if a processor delivers a message m, then all correct processors eventually deliver m.*
3. *Uniform integrity: for any message m, every processor delivers m at most once, and only if m was previously broadcast by its sending processor.*
4. *Gap-free uniform total order: if some processor delivers message m' after message m, then a processor delivers m' only after it has delivered m.*

Key Results

The paper presents three algorithms for solving the problem of atomic broadcast, each under an increasingly demanding failure model, namely, omission, timing, and

Byzantine failures. Each protocol is actually an extension of the previous one.

All three protocols are based on a classical flooding, or information diffusion, algorithm [14]. Every message carries its initiation timestamp T, the name of the initiating processor s, and an update σ. A message is then uniquely identified by (s, T). Then, the basic protocol is simple. Each processor logs every message it receives until it is delivered. When it receives a message that was never seen before, it forwards that message to all other neighbor processors.

Atomic Broadcast for Omission Failures

The first atomic broadcast protocol, supporting omission failures, considers a termination time Δ_o as follows.

$$\Delta_o = \pi\delta + d\delta + \varepsilon \,. \tag{1}$$

The delivery deadline $T + \Delta_o$ is the time by which a processor can be sure that it has received copies of every message with timestamp T (or earlier) that could have been received by some correct process.

The protocol then works as follows. When a processor initiates an atomic broadcast, it propagates that message, similar to the diffusion algorithm described above. The main exception is that every message received after the local clock exceeds the delivery deadline of that message, is discarded. Then, at local time $T + \Delta_o$, a processor delivers all messages timestamped with T, in order of the name of the sending processor. Finally, it discards all copies of the messages from its logs.

Atomic Broadcast for Timing Failures

The second protocol extends the first one by introducing a hop count (i. e., a counter incremented each time a message is relayed) to the messages. With this information, each relaying processor can determine when a message is timely, that is, if a message timestamped T with hop count h is received at time U then the following condition must hold.

$$T - h\varepsilon < U < T + h(\delta + \varepsilon) \,. \tag{2}$$

Before relaying a message, each processor checks the acceptance test above and discard the message if it does not satisfy it. The termination time Δ_t of the protocol for timing failures is as follows.

$$\Delta_t = \pi(\delta + \varepsilon) + d\delta + \varepsilon \,. \tag{3}$$

The authors point out that discarding early messages is not necessary for correctness, but ensures that correct processors keep messages in their log for a bounded amount of time.

Atomic Broadcast for Byzantine Failures

Given some text, every processor is assumed to be able to generate a signature for it, that cannot be faked by other processors. Furthermore, every processor knows the name of every other processors in the network, and has the ability to verify the authenticity of their signature.

Under the above assumptions, the third protocol extends the second one by adding signatures to the messages. To prevent a Byzantine processor (or link) from tampering with the hop count, a message is co-signed by every processor that relays it. For instance, a message signed by k processors $p_1, \ldots, p_k$ is as follows.

$$\big(relayed, \ldots \big(relayed, \big(first, T, \sigma, p_1, s_1\big)\,, p_2, s_2\big)\,,$$
$$\ldots p_k, s_k\big)$$

Where σ is the update, T the timestamp, p_1 the message source, and s_i the signature generated by processor p_i. Any message for which one of the signature cannot be authenticated is simply discarded. Also, if several updates initiated by the same processor p carry the same timestamp, this indicates that p is faulty and the corresponding updates are discarded. The remainder of the protocol is the same as the second one, where the number of hops is given by the number of signatures. The termination time Δ_b is also as follows.

$$\Delta_b = \pi(\delta + \varepsilon) + d\delta + \varepsilon \,. \tag{4}$$

The authors insist however that, in this case, the transmission time δ must be considerably larger than in the previous case, since it must account for the time spent in generating and verifying the digital signatures; usually a costly operation.

Bounds

In addition to the three protocols presented above and their correctness, Cristian *et al.* [7] prove the following two lower bounds on the termination time of atomic broadcast protocols.

Theorem 1 *If the communication network G requires x steps, then any atomic broadcast protocol tolerant of up to π processor and λ link omission failures has a termination time of at least $x\delta + \varepsilon$.*

Theorem 2 *Any atomic broadcast protocol for a Hamiltonian network with n processors that tolerate $n-2$ authentication-detectable Byzantine processor failures cannot have a termination time smaller than* $(n-1)(\delta+\varepsilon)$.

Applications

The main motivation for considering this problem is its use as the cornerstone for ensuring fault-tolerance through process replication. In particular, the authors consider a *synchronous replicated storage*, which they define as a distributed and resilient storage system that displays the same content at every correct physical processor at any clock time. Using atomic broadcast to deliver updates ensures that all updates are applied at all correct processors in the same order. Thus, provided that the replicas are initially consistent, they will remain consistent. This technique, called *state-machine replication* [11,13] or also *active replication*, is widely used in practice as a means of supporting fault-tolerance in distributed systems.

In contrast, Cristian et al. [7] consider atomic broadcast in a *synchronous* system with bounded transmission and processing delays. Their work was motivated by the implementation of a highly-available replicated storage system, with tightly coupled processors running a real-time operating system.

Atomic broadcast has been used as a support for the replication of running processes in real-time systems or, with the problem reformulated to isolate explicit timing requirements, has also been used as a support for fault-tolerance and replication in many group communication toolkits (see survey of Chockler et al. [4]).

In addition, atomic broadcast has been used for the replication of database systems, as a means to reduce the synchronization between the replicas. Wiesmann and Schiper [15] have compared different database replication and transaction processing approaches based on atomic broadcast, showing interesting performance gains.

Cross References

- ▶ Asynchronous Consensus Impossibility
- ▶ Causal Order, Logical Clocks, State Machine Replication
- ▶ Clock Synchronization
- ▶ Failure Detectors

Recommended Reading

1. Carr, R.: The Tandem global update protocol. Tandem Syst. Rev. **1**, 74–85 (1985)
2. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43**, 225–267 (1996)
3. Chang, J.-M., Maxemchuk, N.F.: Reliable broadcast protocols. ACM Trans. Comput. Syst. **2**, 251–273 (1984)
4. Chockler, G., Keidar, I., Vitenberg, R.: Group communication specifications: A comprehensive study. ACM Comput. Surv. **33**, 427–469 (2001)
5. Cristian, F.: Synchronous atomic broadcast for redundant broadcast channels. Real-Time Syst. **2**, 195–212 (1990)
6. Cristian, F., Aghili, H., Strong, R., Dolev, D.: Atomic Broadcast: From simple message diffusion to Byzantine agreement. In: Proc. 15th Intl. Symp. on Fault-Tolerant Computing (FTCS-15), Ann Arbor, June 1985 pp. 200–206. IEEE Computer Society Press
7. Cristian, F., Aghili, H., Strong, R., Dolev, D.: Atomic broadcast: From simple message diffusion to Byzantine agreement. Inform. Comput. **118**, 158–179 (1995)
8. Défago, X., Schiper, A., Urbán, P.: Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Comput. Surveys **36**, 372–421 (2004)
9. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**, 374–382 (1985)
10. Hadzilacos, V., Toueg, S.: Fault-tolerant broadcasts and related problems. In: Mullender, S. (ed.) Distributed Systems, 2nd edn., pp. 97–146. ACM Press Books, Addison-Wesley (1993). Extended version appeared as Cornell Univ. TR 94-1425
11. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Comm. ACM **21**, 558–565 (1978)
12. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Trans. Prog. Lang. Syst. **4**, 382–401 (1982)
13. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surveys **22**, 299–319 (1990)
14. Segall, A.: Distributed network protocols. IEEE Trans. Inform. Theory **29**, 23–35 (1983)
15. Wiesmann, M., Schiper, A.: Comparison of database replication techniques based on total order broadcast. IEEE Trans. Knowl. Data Eng. **17**, 551–566 (2005)

Atomicity

- ▶ Best Response Algorithms for Selfish Routing
- ▶ Linearizability
- ▶ Selfish Unsplittable Flows: Algorithms for Pure Equilibria
- ▶ Snapshots in Shared Memory

Atomic Multicast

- ▶ Atomic Broadcast

Atomic Network Congestion Games

- ▶ Selfish Unsplittable Flows: Algorithms for Pure Equilibria

Atomic Scan

▶ Snapshots in Shared Memory

Atomic Selfish Flows

▶ Best Response Algorithms for Selfish Routing

Attribute-Efficient Learning

1987; Littlestone

Jyrki Kivinen
Department of Computer Science, University of Helsinki,
Helsinki, Finland

Keywords and Synonyms

Learning with irrelevant attributes

Problem Definition

Given here is a basic formulation using the *online mistake bound* model, which was used by Littlestone [9] in his seminal work.

Fix a class C of Boolean functions over n variables. To start a learning scenario, a *target function* $f_* \in C$ is chosen but not revealed to the *learning algorithm*. Learning then proceeds in a sequence of *trials*. At trial t, an input $\boldsymbol{x}_t \in \{0,1\}^n$ is first given to the learning algorithm. The learning algorithm then produces its *prediction* $\hat{y}_t$, which is its guess as to the unknown value $f_*(x_t)$. The correct value $y_t = f_*(x_t)$ is then revealed to the learner. If $y_t \neq \hat{y}_t$, the learning algorithm made a *mistake*. The learning algorithm learns C with mistake bound m, if the number of mistakes never exceeds m, no matter how many trials are made and how f_* and $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ are chosen.

Variable (or attribute) X_i is *relevant* for function $f\colon \{0,1\}^n \to \{0,1\}$ if $f(x_1,\ldots,x_i,\ldots,x_n) \neq f(x_1,\ldots,1-x_i,\ldots,x_n)$ holds for some $\vec{x} \in 0,1^n$. Suppose now that for some $k \leq$ n, every function $f \in C$ has at most k relevant variables. It is said that a learning algorithm learns class C *attribute-efficiently*, if it learns C with a mistake bound polynomial in k and $\log n$. Additionally, the computation time for each trial is usually required to be polynomial in n.

Key Results

The main part of current research of attribute-efficient learning stems from Littlestones Winnow algorithm [9]. The basic version of Winnow maintains a weight vector $\boldsymbol{w}_t = (w_{t,1},\ldots,w_{t,n}) \in \mathbb{R}^n$. The prediction for input $\boldsymbol{x}_t \in \{0,1\}^n$ is given by

$$\hat{y}_t = \operatorname{sign}\left(\sum_{i=1}^{n} w_{t,i}x_{t,i} - \theta\right)$$

where θ is a parameter of the algorithm. Initially $\boldsymbol{w}_1 = (1,\ldots,1)$, and after trial t each component $w_{t,i}$ is updated according to

$$w_{t+1,i} = \begin{cases} \alpha w_{t,i} & \text{if } y_t = 1,\ \hat{y}_t = 0 \text{ and } x_{t,i} = 1 \\ w_{t,i}/\alpha & \text{if } y_t = 0,\ \hat{y}_t = 1 \text{ and } x_{t,i} = 1 \\ w_{t,i} & \text{otherwise} \end{cases} \quad (1)$$

where $\alpha > 1$ is a learning rate parameter.

Littlestone's basic result is that with a suitable choice of θ and α, Winnow learns the class of monotone k-literal disjunctions with mistake bound $O(k \log n)$. Since the algorithm changes its weights only when a mistake occurs, this bound also guarantees that the weights remain small enough for computation times to remain polynomial in n. With simple transformations, Winnow also yields attribute-efficient learning algorithms for general disjunctions and conjunctions. Various subclasses of DNF formulas and decision lists [8] can be learned, too.

Winnow is quite robust against noise, i. e., errors in input data. This is extremely important for practical applications. Remove now the assumption about a target function $f_* \in C$ satisfying $y_t = f_*(x_t)$ for all t. Define *attribute error* of a pair $(\boldsymbol{x}, y)$ with respect to a function f as the minimum Hamming distance between $\boldsymbol{x}$ and $\boldsymbol{x}'$ such that $f(\boldsymbol{x}') = y$. The attribute error of a sequence of trials with respect to f is the sum of attribute errors of the individual pairs $(\boldsymbol{x}_t, y_t)$. Assuming the sequence of trials has attribute error at most A with respect to some k-literal disjunction, Auer and Warmuth [1] show that Winnow makes $O(A + k \log n)$ mistakes. The noisy scenario can also be analyzed in terms of *hinge loss* [5].

The update rule (1) has served as a model for a whole family of *multiplicative update algorithms*. For example, Kivinen and Warmuth [7] introduce the Exponentiated Gradient algorithm, which is essentially Winnow modified for continuous-valued prediction, and show how it can be motivated by a relative entropy minimization principle.

Consider a function class C where each function can be encoded using $O(p(k) \log n)$ bits for some polynomial p. An example would be Boolean formulas with k relevant variables, when the size of the formula is restricted to $p(k)$ ignoring the size taken by the variables. The cardinality of C is then $|C| = 2^{O(p(k) \log n)}$. The classical Halving

Algorithm (see [9] for discussion and references) learns any class consisting of m Boolean functions with mistake bound $\log_2 m$, and would thus provide an attribute-efficient algorithm for such a class C. However, the running time would not be polynomial. Another serious drawback would be that the Halving Algorithm does not tolerate any noise. Interestingly, a multiplicative update similar to (1) has been used in Littlestone and Warmuth's Weighted Majority Algorithm [10], and also Vovk's Aggregating Algorithm [14], to produce a noise-tolerant generalization of the Halving Algorithm.

Attribute-efficient learning has also been studied in other learning models than the mistake bound model, such as Probably Approximately Correct learning [4], learning with uniform distribution [12], and learning with membership queries [3]. The idea has been further developed into learning with a potentially infinite number of attributes [2].

Applications

Attribute-efficient algorithms for simple function classes have a potentially interesting application as a component in learning more complex function classes. For example, any monotone k-term DNF formula over variables $x_1,\ldots,x_n$ can be represented as a monotone k-literal disjunction over 2^n variables z_A, where $z_A = \prod_{i \in A} x_i$ for $A \subseteq \{1, \ldots, n\}$ is defined. Running Winnow with the transformed inputs $z \in \{0,1\}^{2^n}$ would give a mistake bound $O(k \log 2^n) = O(kn)$. Unfortunately the running time would be linear in 2^n, at least for a naive implementation. Khardon et al. [6] provide discouraging computational hardness results for this potential application.

Online learning algorithms have a natural application domain in signal processing. In this setting, the sender emits a true signal y_t at time t, for $t = 1, 2, 3, \ldots$. At some later time $(t + d)$, a receiver receives a signal z_t, which is a sum of the original signal y_t and various echoes of earlier signals $y_{t'}$, $t' < t$, all distorted by random noise. The task is to recover the true signal y_t based on received signals $z_t, z_{t-1}, \ldots, z_{t-l}$ over some time window l. Currently attribute-efficient algorithms are not used for such tasks, but see [11] for preliminary results.

Attribute-efficient learning algorithms are similar in spirit to statistical methods that find sparse models. In particular, statistical algorithms that use L_1 regularization are closely related to multiplicative algorithms such as Winnow and Exponentiated Gradient. In contrast, more classical L_2 regularization leads to algorithms that are not attribute-efficient [13].

Cross References

- ▶ Boosting Textual Compression
- ▶ Learning DNF Formulas

Recommended Reading

1. Auer, P., Warmuth, M.K.: Tracking the best disjunction. Mach. Learn. **32**(2), 127–150 (1998)
2. Blum, A., Hellerstein, L., Littlestone, N.: Learning in the presence of finitely or infinitely many irrelevant attributes. J. Comp. Syst. Sci. **50**(1), 32–40 (1995)
3. Bshouty, N., Hellerstein, L.: Attribute-efficient learning in query and mistake-bound models. J. Comp. Syst. Sci. **56**(3), 310–319 (1998)
4. Dhagat, A., Hellerstein, L.: PAC learning with irrelevant attributes. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, pp 64–74. IEEE Computer Society, Los Alamitos (1994)
5. Gentile, C., Warmuth, M.K.: Linear hinge loss and average margin. In: Kearns, M.J., Solla, S.A., Cohn, D.A. (eds.) Advances in neural information processing systems 11, p. 225–231. MIT Press, Cambridge (1999)
6. Khardon, R., Roth, D., Servedio, R.A.: Efficiency versus convergence of boolean kernels for on-line learning algorithms. J. Artif. Intell. Res. **24**, 341–356 (2005)
7. Kivinen, J., Warmuth, M.K.: Exponentiated gradient versus gradient descent for linear predictors. Inf. Comp. **132**(1), 1–64 (1997)
8. Klivans, A.R. Servedio, R.A.: Toward attribute efficient learning of decision lists and parities. J. Mach. Learn. Res. **7**(Apr), 587–602 (2006)
9. Littlestone, N.: Learning quickly when irrelevant attributes abound: A new linear threshold algorithm. Mach. Learn. **2**(4), 285–318 (1988)
10. Littlestone, N., Warmuth, M.K.: The weighted majority algorithm. Inf. Comp. **108**(2), 212–261 (1994)
11. Martin, R.K., Sethares, W.A., Williamson, R.C., Johnson, Jr., C.R.: Exploiting sparsity in adaptive filters. IEEE Trans. Signal Process. **50**(8), 1883–1894 (2002)
12. Mossel, E., O'Donnell, R., Servedio, R.A.: Learning functions of k relevant variables. J. Comp. Syst. Sci. **69**(3), 421–434 (2004)
13. Ng, A.Y.: Feature selection, L_1 vs. L_2 regularization, and rotational invariance. In: Greiner, R., Schuurmans, D. (eds.) Proceedings of the 21st International Conference on Machine Learning, pp 615–622. The International Machine Learning Society, Princeton (2004)
14. Vovk, V.: Aggregating strategies. In: Fulk, M., Case, J. (eds.) Proceedings of the 3rd Annual Workshop on Computational Learning Theory, p. 371–383. Morgan Kaufmann, San Mateo (1990)

Automated Search Tree Generation

2004; Gramm, Guo, Hüffner, Niedermeier

FALK HÜFFNER
Department of Math and Computer Science,
University of Jena, Jena, Germany

Keywords and Synonyms

Automated proofs of upper bounds on the running time of splitting algorithms

Problem Definition

This problem is concerned with the automated development and analysis of search tree algorithms. Search tree algorithms are a popular way to find optimal solutions to NP-complete problems.[1] The idea is to recursively solve several smaller instances in such a way that at least one branch is a yes-instance if and only if the original instance is. Typically, this is done by trying all possibilities to contribute to a solution certificate for a small part of the input, yielding a small local modification of the instance in each branch.

For example, consider the NP-complete CLUSTER EDITING problem: can a given graph be modified by adding or deleting up to k edges such that the resulting graph is a *cluster graph*, that is, a graph that is a disjoint union of cliques? To give a search tree algorithm for CLUSTER EDITING, one can use the fact that cluster graphs are exactly the graphs that do not contain a P_3 (a path of 3 vertices) as an induced subgraph. One can thus solve CLUSTER EDITING by finding a P_3 and splitting it into 3 branches: delete the first edge, delete the second edge, or add the missing edge. By this characterization, whenever there is no P_3 found, one already has a cluster graph. The original instance has a solution with k modifications if and only if at least one of the branches has a solution with $k - 1$ modifications.

Analysis

For NP-complete problems, the running time of a search tree algorithm only depends on the size of the search tree up to a polynomial factor , which depends on the number of branches and the reduction in size of each branch. If the algorithm solves a problem of size s and calls itself recursively for problems of sizes $s - d_1, \ldots, s - d_i$, then $(d_1, \ldots, d_i)$ is called the *branching vector* of this recursion. It is known that the size of the search tree is then $O(\alpha^s)$, where the *branching number* α is the only positive real root of the *characteristic polynomial*

$$z^d - z^{d-d_1} - \cdots - z^{d-d_i} \, , \qquad (1)$$

where $d = \max\{d_1, \ldots, d_i\}$. For the simple CLUSTER EDITING search tree algorithm and the size measure k, the branching vector is $(1, 1, 1)$ and the branching number is 3, meaning that the running time is up to a polynomial factor $O(3^k)$.

Case Distinction

Often, one can obtain better running times by distinguishing a number of cases of instances, and giving a specialized branching for each case. The overall running time is then determined by the branching number of the worst case. Several publications obtain such algorithms by hand (e. g., a search tree of size $O(2.27^k)$ for CLUSTER EDITING [4]); the topic of this work is how to automate this. That is, the problem is the following:

Problem 1 (Fast Search Tree Algorithm)
INPUT: *An NP-hard problem $\mathcal{P}$ and a size measure s(I) of an instance I of $\mathcal{P}$ where instances I with $s(I) = 0$ can be solved in polynomial time.*
OUTPUT: *A partition of the instance set of $\mathcal{P}$ into cases, and for each case a branching such that the maximum branching number over all branchings is as small as possible.*

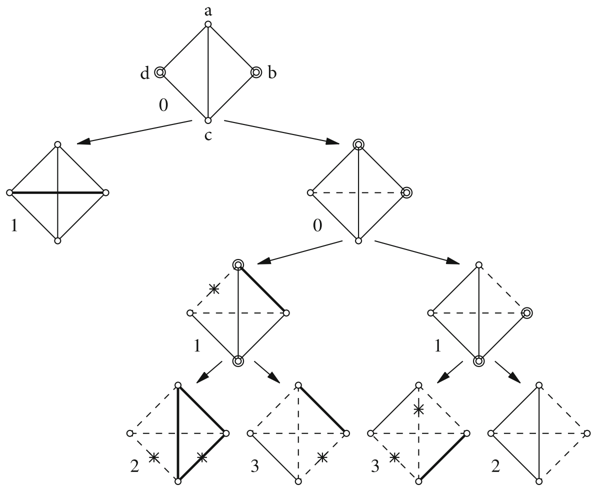
Note that this problem definition is somewhat vague; in particular, to be useful, the case an instance belongs to must be recognizable quickly. It is also not clear whether an optimal search tree algorithm exists; conceivably, the branching number can be continuously reduced by increasingly complicated case distinctions.

Key Results

Gramm et al. [3] describe a method to obtain fast search tree algorithms for CLUSTER EDITING and related problems, where the size measure is the number of editing operations k. To get a case distinction, a number of subgraphs are enumerated such that each instance is known to contain at least one of these subgraphs. It is next described how to obtain a branching for a particular case.

A standard way of systematically obtaining specialized branchings for instance cases is to use a combination of *basic branching* and *data reduction rules*. Basic branching is typically a very simple branching technique, and data reduction rules replace an instance with a smaller, solution-equivalent instance in polynomial time. Applying this to CLUSTER EDITING first requires a small modification of the problem: one considers an *annotated* version, where an edge can be marked as *permanent* and a non-edge can be marked as *forbidden*. Any such annotated vertex pair cannot be edited anymore. For a pair of vertices, the basic branching then branches into two cases: permanent or forbidden (one of these options will require an editing operation). The reduction rules are: if two permanent edges are

[1] For ease of presentation, only decision problems are considered; adaption to optimization problems is straightforward.



Automated Search Tree Generation, Figure 1

Branching for a Cluster Editing case using only basic branching on vertex pairs (*double circles*), and applications of the reduction rules (*asterisks*). Permanent edges are marked *bold*, forbidden edges *dashed*. The *numbers* next to the subgraphs state the change of the problem size k. The branching vector is $(1, 2, 3, 3, 2)$, corresponding to a search tree size of $O(2.27^k)$

adjacent, the third edge of the triangle they induce must also be permanent; and if a permanent and a forbidden edge are adjacent, the third edge of the triangle they induce must be forbidden.

Figure 1 shows an example branching derived in this way.

Using a refined method of searching the space for all possible cases and to distinguish all branchings for a case, Gramm et al. [3] derive a number of search tree algorithms for graph modification problems.

Applications

Gramm et al. [3] apply the automated generation of search tree algorithms to several graph modification problems (see also Table 1). Further, Hüffner [5] demonstrates an application of Dominating Set on graphs with maximum degree 4, where the size measure is the size of the dominating set.

Fedin and Kulikov [2] examine variants of SAT; however, their framework is limited in that it only proves upper bounds for a fixed algorithm instead of generating algorithms.

Skjernaa [6] also presents results on variants of SAT. His framework does not require user-provided data reduction rules, but determines reductions automatically.

Automated Search Tree Generation, Table 1

Summary of search tree sizes where automation gave improvements. "Known" is the size of the best previously published "hand-made" search tree. For the satisfiability problems, m is the number of clauses and l is the length of the formula

Problem	Trivial	Known	New
Cluster Editing	3	2.27	1.92 [3]
Cluster Deletion	2	1.77	1.53 [3]
Cluster Vertex Deletion	3	2.27	2.26 [3]
Bounded Degree Dominating Set	4		3.71 [5]
X3SAT, size measure m	3	1.1939	1.1586 [6]
$(n, 3)$-MaxSAT, size measure m	2	1.341	1.2366 [2]
$(n, 3)$-MaxSAT, size measure l	2	1.1058	1.0983 [2]

Open Problems

The analysis of search tree algorithms can be much improved by describing the "size" of an instance by more than one variable, resulting in multivariate recurrences [1]. It is open to introduce this technique into an automation framework.

It has frequently been reported that better running time bounds obtained by distinguishing a large number of cases do not necessarily speed up, but in fact can slow down, a program. A careful investigation of the tradeoffs involved and a corresponding adaption of the automation frameworks is an open task.

Experimental Results

Gramm et al. [3] and Hüffner [5] report search tree sizes for several NP-complete problems. Further, Fedin and Kulikov [2] and Skjernaa [6] report on variants of satisfiability. Table 1 summarizes the results.

Cross References

► Vertex Cover Search Trees

Acknowledgments

Partially supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4.

Recommended Reading

1. Eppstein, D.: Quasiconvex analysis of backtracking algorithms. In: Proc. 15th SODA, ACM/SIAM, pp. 788–797 (2004)
2. Fedin, S.S., Kulikov, A.S.: Automated proofs of upper bounds on the running time of splitting algorithms. J. Math. Sci. **134**, 2383–2391 (2006). Improved results at http://logic.pdmi.ras.ru/~kulikov/autoproofs.html

3. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Automated generation of search tree algorithms for hard graph modification problems. Algorithmica **39**, 321–347 (2004)
4. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Graph-modeled data clustering: Exact algorithms for clique generation. Theor. Comput. Syst. **38**, 373–392 (2005)
5. Hüffner, F.: Graph Modification Problems and Automated Search Tree Generation. Diplomarbeit, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen (2003)
6. Skjernaa, B.: Exact Algorithms for Variants of Satisfiability and Colouring Problems. Ph. D. thesis, University of Aarhus, Department of Computer Science (2004)