L

Learning with the Aid of an Oracle

1996; Bshouty, Cleve, Gavaldà, Kannan, Tamon

Christino Tamon
Department of Mathematics and Computer Science,
Clarkson University,
Potsdam, NY, USA

Keywords and Synonyms

Oracles and queries that are sufficient for exact learning

Problem Definition

In the exact learning model of Angluin [1], a learning algorithm A must discover an unknown function $\mathsf{f}: \{0,1\}^n \to \{0,1\}$ that is a member of a known class C of Boolean functions. The learning algorithm can make at least one of the following types of queries about f:

- Equivalence query $\mathsf{EQ_f(g)}$, for a candidate function g: The reply is either "yes", if $\mathsf{g} \Leftrightarrow \mathsf{f}$, or a counterexample a with $\mathsf{g}(a) \neq \mathsf{f}(a)$, otherwise.
- Membership query $\mathsf{MQ_f}(a)$, for some $a \in \{0,1\}^n$: The reply is the Boolean value $\mathsf{f}(a)$.
- Subset query $\mathsf{SubQ_f(g)}$, for a candidate function g: The reply is "yes", if $\mathsf{g} \Rightarrow \mathsf{f}$, or a counterexample a with $\mathsf{f}(a) < \mathsf{g}(a)$, otherwise.
- Superset query $\mathsf{SupQ_f(g)}$, for a candidate function g: The reply is "yes", if $\mathsf{f} \Rightarrow \mathsf{g}$, or a counterexample a with $\mathsf{g}(a) < \mathsf{f}(a)$, otherwise.

A Disjunctive Normal Formula (DNF) is a depth-2 OR-AND circuit whose size is given by the number of its AND gates. Likewise, a Conjunctive Normal Formula (CNF) is a depth-2 AND-OR circuit whose size is given by the number of its OR gates. Any Boolean function can be represented as both a DNF or a CNF formula. A k-DNF is a DNF where each AND gate has a fan-in of at most k; similarly, it is possible to define k-CNF.

Problem

For a given class C of Boolean functions, such as polynomial-size Boolean circuits or Disjunctive Normal Form (DNF) formulas, the goal is to design polynomial-time learning algorithms for any unknown $\mathsf{f} \in \mathsf{C}$ and ask a polynomial number of queries. The output of the learning algorithm should be a function g of polynomial size satisfying $g \Leftrightarrow f$. The polynomial functions bounding the running time, query complexity, and output size are defined in terms of the number of inputs n and the size of the smallest representation (Boolean circuit or DNF) of the unknown function f

Key Results

One of the main results proved in [4] is that Boolean circuits and Disjunctive Normal Formulas are exactly learnable using equivalence queries and access to an NP oracle.

Theorem 1 *The following tasks can be accomplished with probabilistic polynomial-time algorithms that have access to an* NP *oracle and make polynomially many equivalence queries:*

- *Learning* DNF *formulas of size s using equivalence queries that are depth-3* AND-OR-AND *formulas of size $O(sn^2/log^2 n)$.*
- *Learning Boolean circuits of size s using equivalence queries that are circuits of size $O(sn + n \log n)$.*

The idea behind this result is simple. Any class C of Boolean functions is exactly learnable with equivalence queries using the Halving algorithm of Littlestone [10]. This algorithm asks equivalence queries that are the *majority* of candidate functions from C. These are functions in C that are consistent with the counterexamples obtained so far by the learning algorithm. Since each such majority query eliminates at least half of the candidate functions, $\log_2 |\mathsf{C}|$ equivalence queries are sufficient to learn any function in C. A problem with using the Halving al-

gorithm here is that the majority query has exponential size. But, it can be shown that a majority of a polynomial number of uniformly random candidate functions is a good enough approximator to the majority of all candidate functions. Moreover, with access to an NP oracle, there is a randomized polynomial time algorithm for generating random uniform candidate functions due to Jerrum, Valiant, and Vazirani [6]. This yields the result.

The next observation is that subset and superset queries are apparently powerful enough to simulate both equivalence queries and the NP oracle. This is easy to see since the tautology test $g \Leftrightarrow 1$ is equivalent to $\mathsf{SubQ}_f(\overline{g}) \wedge \mathsf{SubQ}_f(g)$, for any unknown function f; and, $\mathsf{EQ}_f(g)$ is equivalent to $\mathsf{SubQ}_f(g) \wedge \mathsf{SupQ}_f(g)$. Thus, the following generalization of Theorem 1 is obtained.

Theorem 2 *The following tasks can be accomplished with probabilistic polynomial-time algorithms that make polynomially many subset and superset queries:*

- *Learning* DNF *formulas of size s using equivalence queries that are depth-3* AND-OR-AND *formulas of size* $O(sn^2/log^2 n)$.
- *Learning Boolean circuits of size s using equivalence queries that are circuits of size* $O(sn + n \log n)$.

Stronger deterministic results are obtained by allowing more powerful complexity-theoretic oracles. The first of these results employ techniques developed by Sipser and Stockmeyer [11,12].

Theorem 3 *The following tasks can be accomplished with deterministic polynomial-time algorithms that have access to an* Σ_3^p *oracle and make polynomially many equivalence queries:*

- *Learning* DNF *formulas of size s using equivalence queries that are depth-3* AND-OR-AND *formulas of size* $O(sn^2/log^2 n)$.
- *Learning Boolean circuits of size s using equivalence queries that are circuits of size* $O(sn + n \log n)$.

In the following result, C is an infinite class of functions containing functions of the form $f: \{0,1\}^\star \to \{0,1\}$. The class C is *p*-evaluatable if the following tasks can be performed in polynomial time:

- Given y, is y a valid representation for any function $f_y \in C$?
- Given a valid representation y and $x \in \{0,1\}^\star$, is $f_y(x) = 1$?

Theorem 4 *Let* C *be any p-evaluatable class. The following statements are equivalent:*

- C *is learnable from polynomially many equivalence queries of polynomial size (and unlimited computational power).*
- C *is learnable in deterministic polynomial time with equivalence queries and access to a* Σ_5^p *oracle.*

For exact learning with membership queries, the following results are proved.

Theorem 5 *The following tasks can be accomplished with deterministic polynomial-time algorithms that have access to an* NP *oracle and make polynomially many membership queries (in n,* DNF *and* CNF *sizes of* f*, where* f *is the unknown function):*

- *Learning monotone Boolean functions.*
- *Learning* $O(\log n) -$ CNF $\bigcap O(\log n) -$ DNF.

The ideas behind the above result use techniques from [1,3]. For a monotone Boolean function f, the standard closure algorithm uses both equivalence and membership queries to learn f using candidate functions g satisfying $g \Rightarrow f$. The need for membership can be removed using the following observation. Viewing $\neg f$ as a monotone function on the inverted lattice, it is possible to learn f and $\neg f$ simultaneously using candidate functions g,h, respectively, that satisfy $g \Rightarrow h$. The NP oracle is used to obtain an example a that either helps in learning f or in learning $\neg f$; when no such example can be found, f was learned.

Theorem 6 *Any class* C *of Boolean functions that is exactly learnable using a polynomial number of membership queries (and unlimited computational power) is exactly learnable in expected polynomial time using a polynomial number of membership queries and access to an NP oracle.*

Moreover, any p-evaluatable class C *that is exactly learnable from a polynomially number membership queries (and unlimited computational power), is also learnable in deterministic polynomial time using a polynomial number of membership queries and access to a* Σ_5^p *oracle.*

Theorems 4 and 6 showed that information-theoretic learnability using equivalence and membership queries can be transformed into computational learnability at the expense of using the Σ_5^p and NP oracles, respectively.

Applications

The learning algorithm for Boolean circuits using equivalence queries and access to an NP oracle has found an application in complexity theory. Watanabe (see [9]) showed an improvement on a known theorem of Karp

and Lipton [7]: if NP has polynomial-size circuits, then the polynomial-time hierarchy PH collapses to $\mathrm{ZPP}^{\mathrm{NP}}$.

Some techniques developed in Theorem 5 for exact learning using membership queries of monotone Boolean functions have found applications in data mining [5].

Open Problems

It is unknown if there are polynomial-time learning algorithms for Boolean circuits and DNF formulas using equivalence queries (without complexity-theoretic oracles). There are strong cryptographic evidence that Boolean circuits are not learnable in polynomial-time (see [2] and the references therein). The best running time for learning DNF formulas is $2^{\tilde{O}(n^{1/3})}$ as given by Klivans and Servedio [8]. It is unclear if membership queries help in this case.

Cross References

For related learning results, see ▶ Learning DNF Formulas and ▶ Learning Automata in this encyclopedia.

Recommended Reading

1. Angluin, D.: Queries and Concept Learning. Mach. Learn. **2**, 319–342 (1988)
2. Angluin, D., Kharitonov, M.: When Won't Membership Queries Help? J. Comput. Syst. Sci. **50**, 336–355 (1995)
3. Bshouty, N.H.: Exact Learning Boolean Function via the Monotone Theory. Inform. Comput. **123**, 146–153 (1995)
4. Bshouty, N.H., Cleve, R., Gavaldà, R., Kannan, S., Tamon, C.: Oracles and Queries That Are Sufficient for Exact Learning. J. Comput. Syst. Sci. **52**(3), 421–433 (1996)
5. Gunopolous, D., Khardon, R., Mannila, H., Saluja, S., Toivonen, H., Sharma, R.S.: Discovering All Most Specific Sentences. ACM Trans. Database Syst. **28**, 140–174 (2003)
6. Jerrum, M.R., Valiant, L.G., Vazirani, V.V.: Random Generation of Combinatorial Structures from a Uniform Distribution. Theor. Comput. Sci. **43**, 169–188 (1986)
7. Karp, R.M., Lipton, R.J.: Some Connections Between Nonuniform and Uniform Complexity Classes. In: Proc. 12th Ann. ACM Symposium on Theory of Computing, 1980, pp. 302–309
8. Klivans, A.R., Servedio, R.A.: Learning DNF in Time $2^{\tilde{O}(n^{1/3})}$. J. Comput. Syst. Sci. **68**, 303–318 (2004)
9. Köbler, J., Watanabe, O.: New Collapse Consequences of NP Having Small Circuits. SIAM J. Comput. **28**, 311–324 (1998)
10. Littlestone, N.: Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm. Mach. Learn. **2**, 285–318 (1987)
11. Sipser, M.: A complexity theoretic approach to randomness. In: Proc. 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 330–334
12. Stockmeyer, L.J.: On approximation algorithms for #*P*. SIAM J. Comput. **14**, 849–861 (1985)

Learning Automata

2000; Beimel, Bergadano, Bshouty, Kushilevitz, Varricchio

Amos Beimel[1], Francesco Bergadano[2], Nader H. Bshouty[3], Eyal Kushilevitz[3], Stefano Varricchio[4]

[1] Ben-Gurion University, Beer Sheva, Israel
[2] University of Torino, Torino, Italy
[3] Technion, Haifa, Israel
[4] Department of Computer Science, University of Roma, Rome, Italy

Keywords and Synonyms

Computational learning; Machine learning; Multiplicity automata; Formal series; Boolean formulas; Multivariate polynomials

Problem Definition

This problem is concerned with the learnability of *multiplicity automata* in Angluin's *exact learning model* and applications to the learnability of functions represented by small multiplicity automata.

The Learning Model It is the *exact learning* model [2]: Let f be a *target* function. A learning algorithm may propose to an oracle, in each step, two kinds of queries: *membership queries* (MQ) and *equivalence queries* (EQ). In a MQ it may query for the value of the function f on a particular assignment z. The response to such a query is the value $f(z)$.[1] In a EQ it may propose to the oracle a hypothesis function h. If h is equivalent to f on all input assignments then the answer to the query is YES and the learning algorithm succeeds and halts. Otherwise, the answer to the equivalence query is NO and the algorithm receives a *counterexample*, i. e., an assignment z such that $f(z) \neq h(z)$. One says that the learner *learns* a class of functions C, if for every function $f \in C$ the learner outputs a hypothesis h that is equivalent to f and does so in time polynomial in the "size" of a shortest representation of f and the length of the longest counterexample. The exact learning model is strictly related to the *Probably Approximately Correct* (PAC) model of Valiant [19]. In fact, every equivalence query can be easily simulated by a sample of random examples. Therefore, learnability in the exact learning model also implies learnability in the PAC model with membership queries [2,19].

[1] If f is boolean this is the standard membership query.

Multiplicity Automata Let $\mathcal{K}$ be a field, Σ be an alphabet, ϵ be the empty string. A *multiplicity automaton* (MA) A of size r consists of $|\Sigma|$ matrices $\{\mu_\sigma : \sigma \in \Sigma\}$ each of which is an $r \times r$ matrix of elements from $\mathcal{K}$ and an r-tuple $\vec{\gamma} = (\gamma_1, \ldots, \gamma_r) \in \mathcal{K}^r$. The automaton A defines a function $f_A : \Sigma^* \to \mathcal{K}$ as follows. First, define a mapping μ, which associates with every string in Σ^* an $r \times r$ matrix over $\mathcal{K}$, by $\mu(\epsilon) \triangleq \mathrm{ID}$, where ID denotes the *identity matrix*, and for a string $w = \sigma_1\sigma_2 \ldots \sigma_n$, let $\mu(w) \triangleq \mu_{\sigma_1} \cdot \mu_{\sigma_2} \cdots \mu_{\sigma_n}$. A simple property of μ is that $\mu(x \circ y) = \mu(x) \cdot \mu(y)$, where $\circ$ denotes concatenation. Now, $f_A(w) \triangleq [\mu(w)]_1 \cdot \vec{\gamma}$ (where $[\mu(w)]_i$ denotes the ith row of the matrix $\mu(w)$). Let $f : \Sigma^* \to \mathcal{K}$ be a function. Associate with f an infinite matrix F, where each of its rows is indexed by a string $x \in \Sigma^*$ and each of its columns is indexed by a string $y \in \Sigma^*$. The (x, y) entry of F contains the value $f(x \circ y)$. The matrix F is called the *Hankel Matrix* of f. The xth row of F is denoted by F_x. The (x, y) entry of F may be therefore denoted as $F_x(y)$ and as $F_{x,y}$. The following result relates the size of the minimal MA for f to the rank of F (cf. [4] and references therein).

Theorem 1 *Let $f : \Sigma^* \to \mathcal{K}$ such that $f \not\equiv 0$ and let F be its Hankel matrix. Then, the size r of the smallest multiplicity automaton A such that $f_A \equiv f$ satisfies $r = \mathrm{rank}(F)$ (over the field $\mathcal{K}$).*

Key Results

The learnability of multiplicity automata has been proved in [7] and, independently, in [17]. In what follows let $\mathcal{K}$ be a field, $f : \Sigma^* \to \mathcal{K}$ be a function and F its Hankel matrix such that $r = \mathrm{rank}(F)$ (over $\mathcal{K}$).

Theorem 2 ([4]) *The function f is learnable by an algorithm in time $O(|\Sigma| \cdot r \cdot \mathrm{M}(r) + m \cdot r^3)$ using r equivalence queries and $O((|\Sigma| + \log m)r^2)$ membership queries, where m is the size of the longest counterexample obtained during the execution of the algorithm, and $\mathrm{M}(r)$ is the complexity of multiplying two $r \times r$ matrices.*

Some extensions of the above result can be found in [8,13,16]. In many cases of interest the domain of the target function f is not Σ^* but rather Σ^n for some value n, i. e., $f : \Sigma^n \to \mathcal{K}$. The length of counterexamples, in this case, is always n and so $m = n$. Denote by F^d the submatrix of F whose rows are strings in Σ^d and its columns are strings in Σ^{n-d} and let $r_{\max} = \max_{d=0}^{n} \mathrm{rank}(F^d)$ (where rank is taken over $\mathcal{K}$).

Theorem 3 ([4]) *The function f is learnable by an algorithm in time $O(|\Sigma| rn \cdot \mathrm{M}(r_{\max}))$ using $O(r)$ equivalence queries and $O((|\Sigma| + \log n)r \cdot r_{\max})$ membership queries.*

The time complexity of the two above results has been recently further improved [9].

Applications

The results of this section can be found in [3,4,5,6]. They show the learnability of various classes of functions as a consequence of Theorems 2 and 3. This can be done by proving that for every function f in the class in question, the corresponding Hankel matrix F has low rank. As is well known, any nondeterministic automaton can be regarded as a multiplicity automaton, whose associated function returns the number of accepting paths of the nondeterministic automaton on w. Therefore, the learnability of multiplicity automata gives a new algorithm for learning deterministic automata and unambiguous automata[2]. The learnability of deterministic automata has been proved in [1]. By [14], the class of deterministic automata contains the class of $O(\log n)$-term DNF, i. e., DNF formulae over n boolean variables with $O(\log n)$ number of terms. Hence, this class can be learned using multiplicity automata.

Classes of Polynomials

Theorem 4 *Let $p_{i,j} : \Sigma \to \mathcal{K}$ be arbitrary functions of a single variable ($1 \le i \le t$, $1 \le j \le n$). Let $g_i : \Sigma^n \to \mathcal{K}$ be defined by $\prod_{j=1}^{n} p_{i,j}(z_j)$. Finally, let $f : \Sigma^n \to \mathcal{K}$ be defined by $f = \sum_{i=1}^{t} g_i$. Let F be the Hankel matrix corresponding to f, and F^d the sub-matrices defined in the previous section. Then, for every $0 \le d \le n$, $\mathrm{rank}(F^d) \le t$.*

Corollary 5 *The class of functions that can be expressed as functions over $\mathrm{GF}(p)$ with t summands, where each summand T_i is a product of the form $p_{i,1}(x_1) \cdots p_{i,n}(x_n)$ (and $p_{i,j} : \mathrm{GF}(p) \to \mathrm{GF}(p)$ are arbitrary functions) is learnable in time $\mathrm{poly}(n, t, p)$.*

The above corollary implies as a special case the learnability of polynomials over $\mathrm{GF}(p)$. This extends the result of [18] from multi-linear polynomials to arbitrary polynomials. The algorithm of Theorem 3, for polynomials with n variables and t terms, uses $O(nt)$ equivalence queries and $O(t^2 n \log n)$ membership queries. The special case of the above class – the class of polynomials over $\mathrm{GF}(2)$ – was known to be learnable before [18]. Their algorithm uses $O(nt)$ equivalence queries and $O(t^3 n)$ membership queries. The following theorem extends the latter result to *infinite* fields, assuming that the functions $p_{i,j}$ are bounded-degree polynomials.

[2] A nondeterministic automata is *unambiguous* if for every $w \in \Sigma^*$ there is at most one accepting path.

Theorem 6 *The class of functions over a field $\mathcal{K}$ that can be expressed as t summands, where each summand T_i is of the form $p_{i,1}(x_1)\cdots p_{i,n}(x_n)$, and $p_{i,j}:\mathcal{K}\to\mathcal{K}$ are univariate polynomials of degree at most k, is learnable in time* $\mathrm{poly}(n,t,k)$. *Furthermore, if $|\mathcal{K}|\geq nk+1$ then this class is learnable from membership queries only in time* $\mathrm{poly}(n,t,k)$ *(with small probability of error).*

Classes of Boxes

Let $[\ell]$ denotes the set $\{0,1,\ldots,\ell-1\}$. A box in $[\ell]^n$ is defined by two corners $(a_1,\ldots,a_n)$ and $(b_1,\ldots,b_n)$ (in $[\ell]^n$) as follows:

$$B_{a_1,\ldots,a_n,b_1,\ldots,b_n}=\{(x_1,\ldots,x_n)\colon \forall i,\ a_i\leq x_i\leq b_i\}\,.$$

A box can be represented by its characteristic function in $[\ell]^n$. The following result concerns a more general class of functions.

Theorem 7 *Let $p_{i,j}:\Sigma\to\{0,1\}$ be arbitrary functions of a single variable $(1\leq i\leq t,\ 1\leq j\leq n)$. Let $g_i:\Sigma^n\to\{0,1\}$ be defined by $\prod_{j=1}^{n}p_{i,j}(z_j)$. Assume that there is no point $x\in\Sigma^n$ such that $g_i(x)=1$ for more than s functions g_i. Finally, let $f:\Sigma^n\to\{0,1\}$ be defined by $f=\bigvee_{i=1}^{t}g_i$. Let F be the Hankel matrix corresponding to f. Then, for every field $\mathcal{K}$ and for every $0\leq d\leq n$,* $\mathrm{rank}(F^d)\leq\sum_{i=1}^{s}\binom{t}{i}$.

Corollary 8 *The class of unions of disjoint boxes can be learned in time* $\mathrm{poly}(n,t,\ell)$ *(where t is the number of boxes in the target function). The class of unions of $O(\log n)$ boxes can be learned in time* $\mathrm{poly}(n,\ell)$.

Classes of DNF Formulae

The learnability of DNF formulae has been widely investigated. The following special case of Corollary 5 solves an open problem of [18]:

Corollary 9 *The class of functions that can be expressed as exclusive-OR of t (not necessarily monotone) monomials is learnable in time* $\mathrm{poly}(n,t)$.

While Corollary 9 does not refer to a subclass of DNF, it already implies the learnability of disjoint (i. e., satisfy-1) DNF. Since DNF is a special case of union of boxes (with $\ell=2$), one obtains also the learnability of disjoint DNF from Corollary 8. Positive results for satisfy-s DNF (i. e., DNF formulae in which each assignment satisfies at most s terms) can be obtained, with larger values of s. The following two important corollaries follow from Theorem 7. Note that Theorem 7 holds in any field. For convenience (and efficiency), let $\mathcal{K}=\mathrm{GF}(2)$.

Theorem 10 *Let $f=T_1\vee T_2\vee\cdots\vee T_t$ be a satisfy-s DNF (that is, each T_i is a monomial). Let F be the Hankel matrix corresponding to f. Then,* $\mathrm{rank}(F^d)\leq\sum_{i=1}^{s}\binom{t}{i}\leq t^s$.

Corollary 11 *The class of satisfy-s DNF formulae, for $s=O(1)$, is learnable in time* $\mathrm{poly}(n,t)$.

Corollary 12 *The class of satisfy-s, t-term DNF formulae is learnable in time* $\mathrm{poly}(n)$ *for the following choices of s and t: (1) $t=O(\log n)$; (2) $t=\mathrm{polylog}(n)$ and $s=O(\log n/\log\log n)$; (3) $t=2^{O(\log n/\log\log n)}$ and $s=O(\log\log n)$.*

Classes of Decision Trees

The algorithm of Theorem 3 efficiently learns the class of disjoint DNF formulae. This includes the class of decision trees. Therefore, decision trees of size t on n variables are learnable using $O(tn)$ equivalence queries and $O(t^2n\log n)$ membership queries. This is better than the best known algorithm for decision trees [11] (which uses $O(t^2)$ equivalence queries and $O(t^2n^2)$ membership queries). The following results concern more general classes of decision trees.

Corollary 13 *Consider the class of decision trees that compute functions $f:\mathrm{GF}(p)^n\to\mathrm{GF}(p)$ as follows: each node v contains a query of the form "$x_i\in S_v$?", for some $S_v\subseteq\mathrm{GF}(p)$. If $x_i\in S_v$ then the computation proceeds to the left child of v and if $x_i\notin S_v$ the computation proceeds to the right child. Each leaf ℓ of the tree is marked by a value $\gamma_\ell\in\mathrm{GF}(p)$ which is the output on all the assignments which reach this leaf. Then, this class is learnable in time* $\mathrm{poly}(n,|L|,p)$, *where L is the set of leaves.*

The above result implies the learnability of decision trees with "greater-than" queries in the nodes, solving a problem of [11]. Every decision tree with "greater-than" queries that computes a boolean function can be expressed as the union of disjoint boxes. Hence, this case can also be derived from Corollary 8. The next theorem will be used to learn more classes of decision trees.

Theorem 14 *Let $g_i:\Sigma^n\to\mathcal{K}$ be arbitrary functions $(1\leq i\leq\ell)$. Let $f:\Sigma^n\to\mathcal{K}$ be defined by $f=\prod_{i=1}^{\ell}g_i$. Let F be the Hankel matrix corresponding to f, and G_i be the Hankel matrix corresponding to g_i. Then,* $\mathrm{rank}(F^d)\leq\prod_{i=1}^{\ell}\mathrm{rank}(G_i^d)$.

This theorem has some interesting applications. The first application states that arithmetic circuits of depth two with multiplication gate of fan-in $O(\log n)$ at the top level and addition gates with unbounded fan-in in the bottom level are learnable.

Corollary 15 *Let C be the class of functions that can be expressed in the following way: Let $p_{i,j} : \Sigma \to \mathcal{K}$ be arbitrary functions of a single variable $(1 \le i \le \ell, 1 \le j \le n)$. Let $\ell = O(\log n)$ and $g_i : \Sigma^n \to \mathcal{K}$ $(1 \le i \le \ell)$ be defined by $\sum_{j=1}^{n} p_{i,j}(z_j)$. Finally, let $f : \Sigma^n \to \mathcal{K}$ be defined by $f = \prod_{i=1}^{\ell} g_i$. Then, C is learnable in time* $\mathrm{poly}(n, |\Sigma|)$.

Corollary 16 *Consider the class of decision trees of depth s, where the query at each node v is a boolean function f_v with $r_{\max} \le t$ (as defined in Section "Key Results") such that $(t+1)^s = \mathrm{poly}(n)$. Then, this class is learnable in time* $\mathrm{poly}(n, |\Sigma|)$.

The above class contains, for example, all the decision trees of depth $O(\log n)$ that contain in each node a term or a XOR of a subset of variables as defined in [15] (in this case $r_{\max} \le 2$).

Negative Results

In [4] some limitation of the learnability via the automaton representation has been proved. One can show that the main algorithm does not efficiently learn several important classes of functions. More precisely, these classes contain functions f that have no "small" automaton, i. e., by Theorem 1, the corresponding Hankel matrix F is "large" over every field $\mathcal{K}$.

Theorem 17 *The following classes are not learnable in time polynomial in n and the formula size using multiplicity automata (over any field $\mathcal{K}$): DNF, Monotone DNF, 2-DNF, Read-once DNF, k-term DNF, for $k = \omega(\log n)$, Satisfy-s DNF, for $s = \omega(1)$, Read-j satisfy-s DNF, for $j = \omega(1)$ and $s = \Omega(\log n)$.*

Some of these classes are known to be learnable by other methods, some are natural generalizations of classes known to be learnable as automata ($O(\log n)$-term DNF [11,12,14], and satisfy-s DNF for $s = O(1)$ (Corollary 11)) or by other methods (read-j satisfy-s for $js = O(\log n / \log\log n)$ [10]), and the learnability of some of the others is still an open problem.

Cross References

- ► Learning Constant-Depth Circuits
- ► Learning DNF Formulas

Recommended Reading

1. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**, 87–106 (1987)
2. Angluin, D.: Queries and concept learning. Mach. Learn. **2**(4), 319–342 (1988)
3. Beimel, A., Bergadano, F., Bshouty, N.H., Kushilevitz, E., Varricchio, S.: On the applications of multiplicity automata in learning. In: Proc. of the 37th Annu. IEEE Symp. on Foundations of Computer Science, pp. 349–358, IEEE Comput. Soc. Press, Los Alamitos (1996)
4. Beimel, A., Bergadano, F., Bshouty, N.H., Kushilevitz, E., Varricchio, S.: Learning Functions Represented as Multiplicity Automata. J. ACM **47**, 506–530 (2000)
5. Beimel, A., Kushilevitz, E.: Learning boxes in high dimension. In: Ben-David S. (ed.) 3rd European Conf. on Computational Learning Theory (EuroCOLT '97), Lecture Notes in Artificial Intelligence, vol. 1208, pp. 3–15. Springer, Berlin (1997) Journal version: Algorithmica **22**, 76–90 (1998)
6. Bergadano, F., Catalano, D., Varricchio, S.: Learning sat-k-DNF formulas from membership queries. In: Proc. of the 28th Annu. ACM Symp. on the Theory of Computing, pp. 126–130. ACM Press, New York (1996)
7. Bergadano, F., Varricchio, S.: Learning behaviors of automata from multiplicity and equivalence queries. In: Proc. of 2nd Italian Conf. on Algorithms and Complexity. Lecture Notes in Computer Science, vol. 778, pp. 54–62. Springer, Berlin (1994). Journal version: SIAM J. Comput. **25**(6), 1268–1280 (1996)
8. Bergadano, F., Varricchio, S.: Learning behaviors of automata from shortest counterexamples. In: EuroCOLT '95, Lecture Notes in Artificial Intelligence, vol. 904, pp. 380–391. Springer, Berlin (1996)
9. Bisht, L., Bshouty, N.H., Mazzawi, H.: On Optimal Learning Algorithms for Multiplicity Automata. In: Proc. of 19th Annu. ACM Conf. Comput. Learning Theory, Lecture Notes in Computer Science. vol. 4005, pp. 184–198. Springer, Berlin (2006)
10. Blum, A., Khardon, R., Kushilevitz, E., Pitt, L., Roth, D.: On learning read-k-satisfy-j DNF. In: Proc. of 7th Annu. ACM Conf. on Comput. Learning Theory, pp. 110–117. ACM Press, New York (1994)
11. Bshouty, N.H.: Exact learning via the monotone theory. In: Proc. of the 34th Annu. IEEE Symp. on Foundations of Computer Science, pp. 302–311. IEEE Comput. Soc. Press, Los Alamitos (1993). Journal version: Inform. Comput. **123**(1), 146–153 (1995)
12. Bshouty, N.H.: Simple learning algorithms using divide and conquer. In: Proc. of 8th Annu. ACM Conf. on Comput. Learning Theory, pp. 447–453. ACM Press, New York (1995). Journal version: Computational Complexity, **6**, 174–194 (1997)
13. Bshouty, N.H., Tamon, C., Wilson, D.K.: Learning Matrix Functions over Rings. Algorithmica **22**(1/2), 91–111 (1998)
14. Kushilevitz, E.: A simple algorithm for learning $O(\log n)$-term DNF. In: Proc. of 9th Annu. ACM Conf. on Comput. Learning Theory, pp 266–269, ACM Press, New York (1996). Journal version: Inform. Process. Lett. **61**(6), 289–292 (1997)
15. Kushilevitz, E., Mansour, Y.: Learning decision trees using the Fourier spectrum. SIAM J. Comput. **22**(6), 1331–1348 (1993)
16. Melideo, G., Varricchio, S.: Learning unary output two-tape automata from multiplicity and equivalence queries. In: ALT '98. Lecture Notes in Computer Science, vol. 1501, pp. 87–102. Springer, Berlin (1998)
17. Ohnishi, H., Seki, H., Kasami, T.: A polynomial time learning algorithm for recognizable series. IEICE Transactions on Information and Systems, **E77-D(10)**(5), 1077–1085 (1994)
18. Schapire, R.E., Sellie, L.M.: Learning sparse multivariate polynomials over a field with queries and counterexamples. J. Comput. Syst. Sci. **52**(2), 201–213 (1996)
19. Valiant, L.G.: A theory of the learnable. Commun. ACM **27**(11), 1134–1142 (1984)

Learning Constant-Depth Circuits

1993; Linial, Mansour, Nisan

Rocco Servedio
Department of Computer Science, Columbia University, New York, NY, USA

Keywords and Synonyms

Learning AC^0 circuits

Problem Definition

This problem deals with learning "simple" Boolean functions $f : \{0,1\}^n \to \{-1,1\}$ from uniform random labeled examples. In the basic uniform distribution PAC framework, the learning algorithm is given access to a *uniform random example oracle* $EX(f,U)$ which, when queried, provides a labeled random example $(x, f(x))$ where x is drawn from the uniform distribution U over the Boolean cube $\{0,1\}^n$. Successive calls to the $EX(f,U)$ oracle yield independent uniform random examples. The goal of the learning algorithm is to output a representation of a hypothesis function $h : \{0,1\}^n \to \{-1,1\}$ which with high probability has high accuracy; formally, for any $\epsilon, \delta > 0$, given ϵ and δ the learning algorithm should output an h which with probability at least $1-\delta$ has $\Pr_{x\in U}[h(x) \neq f(x)] \leq \epsilon$.

Many variants of the basic framework described above have been considered. In the *distribution-independent* PAC learning model, the random example oracle is $EX(f,\mathcal{D})$ where $\mathcal{D}$ is an arbitrary (and unknown to the learner) distribution over $\{0,1\}^n$; the hypothesis h should now have high accuracy with respect to $\mathcal{D}$, i. e. with probability $1-\delta$ it must satisfy $\Pr_{x\in\mathcal{D}}[h(x) \neq f(x)] \leq \epsilon$. Another variant that has been considered is when the distribution $\mathcal{D}$ is assumed to be an unknown *product distribution*; such a distribution is defined by n parameters $0 \leq p_1, \ldots, p_n \leq 1$, and a draw from $\mathcal{D}$ is obtained by independently setting each bit x_i to 1 with probability p_i. Yet another variant is to consider learning with the help of a *membership oracle*: this is a "black-box" oracle $MQ(f)$ for f which, when queried on an input $x \in \{0,1\}^n$, returns the value of $f(x)$. The model of uniform distribution learning with a membership oracle has been well studied, see e. g. [4,11].

There are many ways to make precise the notion of a "simple" Boolean function; one common approach is to stipulate that the function be computed by a Boolean circuit of some restricted form. A circuit of *size s and depth d* consists of s AND and OR gates (of unbounded fanin) in which the longest path from any input literal $x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n$ to the output node is of length d. Note that a circuit of size s and depth 2 is simply a CNF formula or DNF formula. The complexity class consisting of those Boolean functions computed by poly(n)-size, $O(1)$-depth circuits is known as *nonuniform* AC^0.

Key Results

Positive Results

Linial et al. [12] showed that almost all of the "Fourier weight" of any constant-depth circuit is on low-degree Fourier coefficients:

Lemma 1 *Let $f : \{0,1\}^n \to \{-1,1\}$ be a Boolean function that is computed by a circuit of size s and depth d. Then for any integer $t \geq 0$,*

$$\sum_{S\subset\{1,\ldots,n\},|S|>t} \hat{f}(S)^2 \leq 2s2^{-t^{1/d}/20}.$$

(Hastad [3] has given a refined version of Lemma 1 with slightly sharper bounds; see also [17] for a streamlined proof.) They also showed that any Boolean function can be well approximated by approximating its Fourier spectrum:

Lemma 2 *Let $f : \{0,1\}^n \to \{-1,1\}$ be any Boolean function and let $g : \{0,1\}^n \to \boldsymbol{R}$ be an arbitrary function such that $\sum_{S\subseteq\{1,\ldots,n\}}(\hat{f}(S) - \hat{g}(S))^2 \leq \epsilon$. Then $\Pr_{x\in U}[f(x) \neq sign(g(x))] \leq \epsilon$.*

Using the above two results together with a procedure that estimates all the "low-order" Fourier coefficients, they obtained a quasipolynomial-time algorithm for learning constant-depth circuits:

Theorem 3 *There is an $n^{(O(\log(n/\epsilon)))^d}$-time algorithm that learns any poly(n)-size, depth-d Boolean circuit to accuracy ϵ with respect to the uniform distribution, using uniform random examples only.*

Furst et al. [2] extended this result to learning under constant-bounded product distributions. A product distribution $\mathcal{D}$ is said to be *constant-bounded* if each of its n parameters $p_1,\cdots,p_n$ is bounded away from 0 and 1, i. e. satisfies $\min\{p_i, 1-p_i\} = \Theta(1)$.

Theorem 4 *There is an $n^{(O(\log(n/\epsilon)))^d}$-time algorithm that learns any poly(n)-size, depth-d Boolean circuit to accuracy ϵ given random examples drawn from any constant-bounded product distribution.*

By combining the Fourier arguments of Linial et al. with hypothesis boosting, Jackson et al. [5] were able to extend

Theorem 3 to a broader class of circuits, namely constant-depth AND/OR circuits that additionally contain (a limited number of) *majority gates*. A majority gate over r Boolean inputs is a binary gate which outputs "true" if and only if at least half of its r Boolean inputs are set to "true".

Theorem 5 *There is an $n^{\log^{O(1)}(n/\epsilon)}$-time algorithm that learns any poly(n)-size, constant-depth Boolean circuit that contains polylog(n) many majority gates to accuracy ϵ with respect to the uniform distribution, using uniform random examples only.*

Negative Results

Kharitonov [7] showed that under a strong but plausible cryptographic assumption, the algorithmic result of Theorem 3 is essentially optimal. A *Blum integer* is an integer $N = P \cdot Q$ where both P and Q are congruent to 3 modulo 4. Kharitonov proved that if the problem of factoring a randomly chosen n-bit Blum integer is 2^{n^ϵ}-hard for some fixed $\epsilon > 0$, then any algorithm that (even weakly) learns polynomial-size depth-d circuits must run in time $2^{\log^{\Omega(d)} n}$, even if it is only required to learn under the uniform distribution and can use a membership oracle. This implies that there is no polynomial-time algorithm for learning polynomial-size, depth-d circuits (for d larger than some absolute constant).

Using a cryptographic construction of Naor and Reingold [14], Jackson et al. [5] proved a related result for circuits with majority gates. They showed that under Kharitonov's assumption, any algorithm that (even weakly) learns depth-5 circuits consisting of $\log^k n$ many majority gates must run in time $2^{\log^{\Omega(k)} n}$ time, even if it is only required to learn under the uniform distribution and can use a membership oracle.

Applications

The technique of learning by approximating most of the Fourier spectrum (Lemma 2 above) has found many applications in subsequent work on uniform distribution learning. It is a crucial ingredient in the current state-of-the-art algorithms for learning monotone DNF formulas [16], monotone decision trees [15], and intersections of halfspaces [8] from uniform random examples only. Combined with a membership-oracle based procedure for identifying large Fourier coefficients, this technique is at the heart of an algorithm for learning decision trees [11]; this algorithm in turn plays a crucial role in the celebrated polynomial-time algorithm of Jackson [4] for learning polynomial-size depth-2 circuits under the uniform distribution.

The ideas of Linial et al. have also been applied for the difficult problem of *agnostic learning*. In the agnostic learning framework there is a joint distribution $\mathcal{D}$ over example-label pairs $\{0,1\}^n \times \{-1,1\}$; the goal of an agnostic learning algorithm for a class C of functions is to construct a hypothesis h such that $\Pr_{(x,y)\in\mathcal{D}}[h(x) \neq y] \leq \min_{f\in C} \Pr_{(x,y)\in\mathcal{D}}[f(x) \neq y] + \epsilon$. Kalai et al. [6] gave agnostic learning algorithms for halfspaces and related classes via an algorithm which may be viewed as a generalization of Linial et al.'s algorithm to a broader class of distributions.

Finally, there has been some applied work on learning using Fourier representations as well [13].

Open Problems

Perhaps the most outstanding open question related to this work is whether polynomial-size circuits of depth two – i. e. DNF formulas – can be learned in polynomial time from uniform random examples only. Blum [1] has offered a cash prize for a solution to a restricted version of this problem. A hardness result for learning DNF would also be of great interest; recent work of Klivans and Sherstov [10] gives a hardness result for learning ANDs of majority gates, but hardness for DNF (ANDs of ORs) remains an open question.

Another open question is whether the quasipolynomial-time algorithms for learning constant-depth circuits under uniform distributions and product distributions can be extended to the general distribution-independent model. Known results in complexity theory imply that quasipolynomial-time distribution-independent learning algorithms for constant-depth circuits would follow from the existence of efficient linear threshold learning algorithms with a sufficiently high level of tolerance to "malicious" noise. Currently no nontrivial distribution-independent algorithms are known for learning circuits of depth 3; for depth-2 circuits the best known running time in the distribution-independent setting is the $2^{\tilde{O}(n^{1/3})}$-time algorithm of Klivans and Servedio [9].

A third direction for future work is to extend the results of [5] to a broader class of circuits. Can constant-depth circuits augmented with MOD_p gates, or with *weighted* majority gates, be learned in quasipolynomial time? [5] discusses the limitations of current techniques to address these extensions.

Cross References

- ▶ Cryptographic Hardness of Learning
- ▶ Learning DNF Formulas
- ▶ PAC Learning
- ▶ Statistical Query Learning

Recommended Reading

1. Blum, A.: Learning a function of r relevant variables (open problem). In: Proceedings of the 16th Annual Conference on Learning Theory, pp. 731–733, Washington, 24–27 August 2003
2. Furst, M., Jackson, J., Smith, S.: Improved learning of AC^0 functions. In: Proceedings of the Fourth Annual Workshop on Computational Learning Theory, pp. 317–325, Santa Cruz, (1991)
3. Håstad, J.: A slight sharpening of LMN. J. Comput. Syst. Sci. **63**(3), 498–508 (2001)
4. Jackson, J.: An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. J. Comput. Syst. Sci. **55**, 414–440 (1997)
5. Jackson, J., Klivans, A., Servedio, R.: Learnability beyond AC^0. In: Proceedings of the 34th ACM Symposium on Theory of Computing, pp. 776–784, Montréal, 23–25 May 2002
6. Kalai, A., Klivans, A., Mansour, Y., Servedio, R.: Agnostically learning halfspaces. In: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 11–20, Pittsburgh, PA, USA, 23–25 October 2005
7. Kharitonov, M.: Cryptographic hardness of distribution-specific learning. In: Proceedings of the 25th Annual Symposium on Theory of Computing, pp. 372–381. (1993)
8. Klivans, A., O'Donnell, R., Servedio, R.: Learning intersections and thresholds of halfspaces. J. Comput. Syst. Sci. **68**(4), 808–840 (2004)
9. Klivans, A., Servedio, R.: Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. J. Comput. Syst. Sci. **68**(2), 303–318 (2004)
10. Klivans, A., Sherstov, A.: Cryptographic hardness results for learning intersections of halfspaces. In: Proceedings of the 47th Annual Symposium on Foundations of Computer Science, pp. 553–562, Berkeley, 22–24 October 2006
11. Kushilevitz, E., Mansour, Y.: Learning decision trees using the Fourier spectrum. SIAM J. Comput. **22**(6), 1331–1348 (1993)
12. Linial, N., Mansour, Y., Nisan, N.: Constant depth circuits, Fourier transform and learnability. J. ACM **40**(3), 607–620 (1993)
13. Mansour, Y., Sahar, S.: Implementation Issues in the Fourier Transform Algorithm. Mach. Learn. **40**(1), 5–33 (2000)
14. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. J. ACM **51**(2), 231–262 (2004)
15. O'Donnell, R., Servedio, R.: Learning monotone decision trees in polynomial time. In: Proceedings of the 21st Conference on Computational Complexity (CCC), pp. 213–225, Prague, 16–20 July 2006
16. Servedio, R.: On learning monotone DNF under product distributions. Inform Comput **193**(1), 57–74 (2004)
17. Stefankovic, D.: Fourier transforms in computer science. Masters thesis, TR-2002-03, University of Chicago (2002)

Learning DNF Formulas

1997; Jackson

Jeffrey C. Jackson
Department of Mathematics and Computer Science, Duquesne University, Pittsburgh, PA, USA

Keywords and Synonyms

Sum of products notation; Learning disjunctive normal form formulas (or expressions); Learning sums of products

Problem Definition

A Disjunctive Normal Form (DNF) expression is a Boolean expression written as a disjunction of *terms*, where each term is the conjunction of Boolean variables that may or may not be negated. For example, $(v_1 \wedge \overline{v_2}) \vee (v_2 \wedge v_3)$ is a two-term DNF expression over three variables. DNF expressions occur frequently in digital circuit design, where DNF is often referred to as sum of products notation. From a learning perspective, DNF expressions are of interest because they provide a natural representation for certain types of expert knowledge. For example, the conditions under which complex tax rules apply can often be readily represented as DNFs. Another nice property of DNF expressions is their universality: every n-bit Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ can be represented as a DNF expression F over at most n variables.

Informally, the problem to be addressed is the following. A learning algorithm is given access to an oracle $MEM(f)$ that, for some fixed integer $n > 0$, on an n-bit input will return a 1-bit response. The output of the oracle is determined by an n-bit Boolean function f that can be represented by an s-term DNF expression F over n variables. All that is known about f and F is n. The algorithm's goal is to produce, with high probability over the random choices it makes, an n-bit Boolean function h that agrees with f on all but a small fraction of the 2^n elements of the domain of f and h. Furthermore, the algorithm must run in time polynomial in n and s (and other parameters given in the formal problem definition). The algorithm is not required to output a DNF representation of the approximating function h, but h must be computable in time comparable to that needed to evaluate f. In particular, $h(x)$ should be computable in time polynomial in n and s for all $x \in \{0,1\}^n$.

In the following formal problem definition, U_n represents the uniform distribution over $\{0,1\}^n$.

Problem 1 (UDNFL)

Input: *Positive integer n; $\varepsilon, \delta > 0$; oracle $MEM(f)$ for $f : \{0,1\}^n \to \{0,1\}$ expressible as DNF with s terms over n variables.*

Output: *With probability at least $1 - \delta$ over the random choices made by the algorithm, a function $h\colon \{0,1\}^n \to \{0,1\}$ (not necessarily a DNF expression) such that*

$\Pr_{x\sim U_n}[h(x) \neq f(x)] < \varepsilon$. *The algorithm must run in time polynomial in n, s,* $1/\varepsilon$, *and* $1/\delta$, *and for all* $x \in \{0,1\}^n$, *h(x) must be computable in time polynomial in n and s.*

Threshold of Parities (TOP) is another interesting universal representation for Boolean functions. For a and x in $\{0,1\}^n$, the *even parity function* $e_a(x)$ returns 1 if the dot product $a \cdot x$ is even and 0 otherwise. That is, the output is 1 if the parity of the bits in x indexed by a is even and 0 otherwise. Similarly, define the *odd parity function* $o_a(x)$ to return 1 if the parity of the bits in x indexed by a is odd and 0 otherwise. A *parity function* is either an even or an odd parity function. Then a *TOP representation of size s* is defined by a collection of s parity functions $(p_1, p_2, \ldots, p_s)$, where p_i is allowed to be the same as p_j for $i \neq j$ (i. e., there may be fewer than s distinct functions in the collection). The value of a TOP F on input x is the majority value of $p_i(x)$ over the s parity functions defining F (with value 0 in the case of no majority).

Problem 2 (UTOPL)
Input: *Positive integer n;* $\varepsilon, \delta > 0$; *oracle MEM(f) for* $f : \{0,1\}^n \to \{0,1\}$ *expressible as TOP of size s over n variables.*
Output: *With probability at least* $1-\delta$ *over the random choices made by the algorithm, a function* $h : \{0,1\}^n \to \{0,1\}$ *(not necessarily a TOP) such that* $\Pr_{x\sim U_n}[h(x) \neq f(x)] < \varepsilon$. *The algorithm must run in time polynomial in n, s,* $1/\varepsilon$, *and* $1/\delta$, *and for all* $x \in \{0,1\}^n$, *h(x) must be computable in time polynomial in n and s.*

TOP and DNF representations of the same Boolean function f can be related as follows. For every $f : \{0,1\}^n \to \{0,1\}$, if f can be represented by an s-term DNF expression then there is a TOP representation of f of size $O(ns^2)$ [7]. On the other hand, a DNF of size 2^{n-1} is required to represent the parity function $e_{1^n}()$ (even parity in which all bits are relevant). So the DNF expression for a function is at most polynomially more succinct than the optimal equivalent TOP expression, whereas TOP expressions may be exponentially more succinct than the optimal equivalent DNF expressions.

From a learning viewpoint, this means that UTOPL is a harder problem then UDNFL. This is because the only difference between the problems is in how the size parameter s is defined, with larger values of s allowing the learning algorithm more time. Thus, since the DNF size of a function f is never much smaller than the TOP size of f and may be much larger, the learning algorithm is effectively allowed more time for DNF learning than it is for TOP learning.

Another direction in which the DNF problem can naturally be extended is to non-Boolean inputs. A DNF with s terms can be viewed as a union of s subcubes of the Boolean hypercube $\{0,1\}^n$, with the portion of the hypercube covered by this union corresponding to the 1 values of the DNF. Similarly, for any fixed positive integer b, a function $f : \{0,1,\ldots,b-1\}^n \to \{0,1\}$ can be defined as a union of rectangles over $\{0,1,\ldots,b-1\}^n$, where a *rectangle* is the set of all elements in a Cartesian product $\prod_{i=1}^n \{\ell_i, \ell_i+1, \ldots, u_i\}$, where for all i, $0 \leq \ell_i \leq u_i < b$. As in the Boolean case, the 1 values of f correspond exactly to those inputs included in this union of rectangles. Such a representation of f as a union of s rectangles will be called a *UBOX of size s*. Defining U_n^b to be the uniform distribution over $\{0,1,\ldots,b-1\}^n$, this gives rise to the following problem:

Problem 3 (UUBOXL) Input: *Positive integers n and b;* $\varepsilon, \delta > 0$; *oracle MEM(f) for* $f : \{0,1,\ldots,b-1\}^n \to \{0,1\}$ *expressible as UBOX of size s over n variables.*

Output: *With probability at least* $1-\delta$ *over the random choices made by the algorithm, a function* $h : \{0,1,\ldots,b-1\}^n \to \{0,1\}$ *(not necessarily a UBOX) such that* $\Pr_{x\sim U_n^b}[h(x) \neq f(x)] < \varepsilon$. *The algorithm must run in time polynomial in n, s,* $1/\varepsilon$, *and* $1/\delta$, *and for all* $x \in \{0,1\}^n$, *h(x) must be computable in time polynomial in n and s (b is taken to be a constant).*

Key Results

Theorem 1 *There is an algorithm—the Harmonic Sieve[7,9,10]—that solves both* UDNFL *and* UTOPL.

The run time[1] of the original version of the Harmonic Sieve [7] is $\tilde{O}(ns^{10}/\varepsilon^{12+c})$, where c is an arbitrarily small positive constant and the $\tilde{O}()$ notation is the same as big-O notation except that logarithmic factors are suppressed (in particular, the run-time dependence on $1/\delta$ is logarithmic). This bound was improved in [4] and [11] to $\tilde{O}(ns^6/\varepsilon^2)$, and Feldman [6] has further improved the run time to $\tilde{O}(ns^4/\varepsilon)$. All of the improvements use the same overall algorithmic structure as the Harmonic Sieve, but some components of the structure are replaced with more efficient approaches. The output h of the original Sieve is a TOP, but this is not the case for the more efficient versions of the algorithm.

Learnability of TOP implies learnability of several other classes that are, for purposes of uniform learning, special cases of TOP [7]. This includes the class of those functions that can be defined as a majority of arbitrary $(\log n)$-bit Boolean functions (size measure is the number

[1] See [4] for an explanation of an error in the time bound given in [7]

of functions) and the class of those functions that can be expressed as a *parity-DNF*, that is, as an OR of ANDs of parity functions (size measure is the number of ANDs).

Theorem 2 *A variation of the Harmonic Sieve solves* UUBOXL.

An algorithm for UUBOXL is given in [7].

Applications

An extended version of the Harmonic Sieve can tolerate false responses by the oracle $MEM(f)$. In particular, in the *uniform persistent classification noise* learning model, a constant noise rate $0 \leq \eta < 1/2$ is fixed and an oracle $MEM^{\eta}(f)$ is defined as follows: if the query x has not been presented to the oracle previously, then $MEM^{\eta}(f)$ returns $f(x)$ with probability $1-\eta$ and the complement $\overline{f(x)}$ with probability η. If x has been presented to the oracle previously, then it returns the same response that it did on the first query with x. Jackson et al. [8] showed how to modify the Harmonic Sieve to efficiently learn DNF (and TOP, although the referenced paper does not state this) in the uniform persistent classification noise model.

Bshouty and Jackson [3] defined a *uniform quantum example oracle* $QEX(f, U_n)$ that, for a fixed unknown function $f : \{0,1\}^n \to \{0,1\}$, produces a quantum superposition of the 2^n labeled-example pairs $\langle x, f(x)\rangle$, one pair for each $x \in \{0,1\}^n$. All pairs have the same amplitude, $2^{-n/2}$. Bshouty and Jackson then showed that such an oracle $QEX(f, U_n)$ cannot simulate the oracle $MEM(f)$ used by the Harmonic Sieve to learn DNF and TOP, and therefore is a weaker form of oracle. Nevertheless, building on the Harmonic Sieve, they gave an efficient quantum algorithm for learning DNF and TOP from a uniform quantum example oracle.

Bshouty et al. [5] defined a model of learning from uniform random walks over $\{0,1\}^n$. Unlike the oracle $MEM(f)$, where the learning algorithm actively selects examples to be queried, in the random walk model the learner passively accepts random examples from the random walk oracle. Building in part on the Harmonic Sieve, Bshouty et al. showed that DNF is efficiently learnable in the uniform random walk model. In addition, for fixed $0 \leq \rho \leq 1$, Bshouty et al. defined a *ρ-Noise Sensitivity example oracle* $NS - EX_{\rho}(f)$ that, when invoked, selects an input $x \in \{0,1\}^n$ uniformly at random, forms an input y by flipping each bit of x independently at random with probability $\frac{1}{2}(1-\rho)$, and returns the quadruple $\langle x, f(x), y, f(y)\rangle$. This oracle is shown to be no more powerful than the uniform random walk oracle, and an algorithm based in part on the Sieve is presented that, for any constant $\rho \in [0,1]$, efficiently learns DNF using $NS - EX_{\rho}(f)$. However, the question of whether or not TOP is efficiently learnable in either of these models is left open.

Atici and Servedio [1] have given a generalized version of the Harmonic Sieve that can, among other things, learn an interesting subset of the class of unions of rectangles over $\{0, 1, \ldots, b-1\}^n$ for non-constant b.

Open Problems

A key open problem involves relaxing the power of the oracle used. For instance, given a function $f : \{0,1\}^n \to \{0,1\}$, a *uniform example oracle for* f $EX(f, U_n)$ is an oracle that, on every query, randomly selects an input x according to U_n and returns the value $f(x)$. If the definition of UDNFL is changed so that $EX(f, U_n)$ is provided rather than $MEM(f)$, is UDNFL still solvable? There is at least some reason to believe that the answer is no (see, e. g., [2]). The apparently simpler question of whether or not the class of *monotone DNF expressions* (DNF expressions with no negated variables) is efficiently uniform learnable from an example oracle is also still open, and there is less reason to doubt that an algorithm solving this problem will be discovered.

Cross References

- ▶ Learning Constant-Depth Circuits
- ▶ Learning Heavy Fourier Coefficients of Boolean Functions
- ▶ PAC Learning

Recommended Reading

1. Atici, A., Servedio, R.A.: Learning unions of $\omega(1)$-dimensional rectangles. In: Proceedings of 17th Algorithmic Learning Theory Conference, pp. 32–47. Springer, New York (2006)
2. Blum, A., Furst, M., Jackson, J., Kearns, M., Mansour, Y., Rudich, S.: Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In: Proceedings of the 26th Annual ACM Symposium on Theory of Computing, pp. 253–262. Association for computing Machinery, New York (1994)
3. Bshouty, N.H., Jackson, J.C.: Learning DNF over the uniform distribution using a quantum example oracle. SIAM J. Comput. **28**, 1136–1153 (1999)
4. Bshouty, N.H., Jackson, J.C., Tamon, C.: More efficient PAC-learning of DNF with membership queries under the uniform distribution. J. Comput. Syst. Sci. **68**, 205–234 (2004)
5. Bshouty, N.H., Mossel, E., O'Donnell, R., Servedio, R.A.: Learning DNF from random walks. J. Comput. Syst. Sci. **71**, 250–265 (2005)
6. Feldman, V.: On attribute efficient and non-adaptive learning of parities and DNF expressions. In: 18th Annual Conference on Learning Theory, pp. 576–590. Springer-Verlag, Berlin Heidelberg (2005)

7. Jackson, J.: An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. J. Comput. Syst. Sci. **55**, 414–440 (1997)
8. Jackson, J., Shamir, E., Shwartzman, C.: Learning with queries corrupted by classification noise. Discret. Appl. Math. **92**, 157–175 (1999)
9. Jackson, J.C.: An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. In: 35th Annual Symposium on Foundations of Computer Science, pp. 42–53. IEEE Computer Society Press, Los Alamitos (1994)
10. Jackson, J.C.: The Harmonic Sieve: A Novel Application of Fourier Analysis to Machine Learning Theory and Practice. Ph. D. thesis, Carnegie Mellon University (1995)
11. Klivans, A.R., Servedio, R.A.: Boosting and hard-core set construction. Mach. Learn. **51**, 217–238 (2003)

Learning Heavy Fourier Coefficients of Boolean Functions

1989; Goldreich, Levin

Luca Trevisan
Department of Computer Science, University of California at Berkeley, Berkeley, CA, USA

Keywords and Synonyms

Error-control codes, Reed–Muller code

Problem Definition

The Hamming distance $d_H(y,z)$ between two binary strings y and z of the same length is the number of entries in which y and z disagree. A binary error-correcting code of minimum distance d is a mapping $C : \{0,1\}^k \to \{0,1\}^n$ such that for every two distinct inputs $x, x' \in \{0,1\}^k$, the encodings $C(x)$ and $C(x')$ have Hamming distance at least d. Error-correcting codes are employed to transmit information over noisy channels. If a sender transmits an encoding $C(x)$ of a message x via a noisy channel, and the recipient receives a corrupt bit string $y \neq C(x)$, then, provided that y differs from $C(x)$ in at most $(d-1)/2$ locations, the recipient can recover y from $C(x)$. The recipient can do so by searching for the string x that minimizes the Hamming distance between $C(x)$ and y: there can be no other string x' such that $C(x')$ has Hamming distance $(d-1)/2$ or smaller from y, otherwise $C(x)$ and $C(x')$ would be within Hamming distance $d-1$ or smaller, contradicting the above definition. The problem of recovering the message x from the corrupted encoding y is the *unique decoding* problem for the error-correcting code C. For the above-described scheme to be feasible, the decoding problem must be solvable via an efficient algorithm. These notions are due to Hamming [4].

Suppose that C is a code of minimum distance d, and such that there are pairs of encodings $C(x)$, $C(x')$ whose distance is exactly d. Furthermore, suppose that a communication channel is used that could make a number of errors larger than $(d-1)/2$. Then, if the sender transmits an encoded message using C, it is no longer possible for the recipient to uniquely reconstruct the message. If the sender, for example, transmits $C(x)$, and the recipient receives a string y that is at distance $d/2$ from $C(x)$ and at distance $d/2$ from $C(x')$, then, from the perspective of the recipient, it is equally likely that the original message was x or x'. If the recipient knows an upper bound e on the number of entries that the channel has corrupted, then, given the received string y, the recipient can at least compute the list of all strings x such that $C(x)$ and y differ in at most e locations. An error-correcting code $C : \{0,1\}^k \to \{0,1\}^n$ is (e,L)-list decodable if, for every string $y \in \{0,1\}^n$, the set $\{x \in \{0,1\}^k : d_H(C(x), y) \leq e\}$ has cardinality at most L. The problem of reconstructing the list given y and e is the *list-decoding problem* for the code C. Again, one is interested in efficient algorithms for this problem. The notion of list-decoding is due to Elias [1].

A code $C : \{0,1\}^k \to \{0,1\}^n$ is a *Hadamard code* if every two encodings $C(x)$, $C(x')$ differ in precisely $n/2$ locations. In the Computer Science literature, it is common to use the term Hadamard code for a specific construction (the *Reed–Muller code of order 2*) that satisfies the above property. For a string $a \in \{0,1\}^k$, define the function $\ell_a : \{0,1\}^k \to \{0,1\}$ as

$$\ell_a(x) := \sum_i a_i x_i \bmod 2 .$$

Observe that, for $a \neq b$, the two functions ℓ_a and ℓ_b differ on precisely $(2^k)/2$ inputs. For $n = 2^k$, the code $H : \{0,1\}^k \to \{0,1\}^n$ maps a message $a \in \{0,1\}^k$ into the n-bit string which is the *truth-table of the function* ℓ_a. That is, if $b_1, \ldots, b_n$ is an enumeration of the $n = 2^k$ elements of $\{0,1\}^k$, and $a \in \{0,1\}^k$ is a message, then the encoding $H(a)$ is the n-bit string that contains the value $\ell_a(b_i)$ in the i-th entry. Note that any two encodings $H(x)$, $H(x')$ differ in precisely $n/2$ entries, and so what was just defined is a Hadamard code. From now on, the term *Hadamard code* will refer exclusively to this construction.

It is known that the Hadamard code $H : \{0,1\}^k \to \{0,1\}^{2^k}$ is $(\frac{1}{2} - \epsilon, \frac{1}{4\epsilon^2})$-list decodable for every $\epsilon > 0$. The Goldreich–Levin results provide efficient list-decoding algorithm.

The following definition of the *Fourier spectrum* of a boolean function will be needed later to state an application of the Goldreich–Levin results to computational learning theory. For a string $a \in \{0,1\}^k$, define the function $\chi_a : \{0,1\}^k \to \{-1,+1\}$ as $\chi_a(x) := (-1)^{\ell_a(x)}$. Equivalently, $\chi_a(x) = (-1)^{\sum_i a_i x_i}$. For two functions $f, g : \{0,1\}^k \to \mathbb{R}$, define their *inner product* as

$$\langle f, g \rangle := \frac{1}{2^k} \sum_x f(x) \cdot g(x) .$$

Then it is easy to see that, for every $a \neq b$, $\langle \chi_a, \chi_b \rangle = 0$, and $\langle \chi_a, \chi_a \rangle = 1$. This means that the functions $\{\chi_a\}_{a \in \{0,1\}^k}$ form an orthonormal basis for the set of all functions $f : \{0,1\}^k \to \mathbb{R}$. In particular, every such function f can be written as a linear combination

$$f(x) = \sum_a \hat{f}(a) \chi_a(x)$$

where the coefficients $\hat{f}(a)$ satisfy $\hat{f}(a) = \langle f, \chi_a \rangle$. The coefficients $\hat{f}(a)$ are called the *Fourier coefficients* of the function f.

Key Results

Theorem 1 *There is a randomized algorithm GL that, given in input an integer k and a parameter $\epsilon > 0$, and given oracle access to a function $f : \{0,1\}^k \to \{0,1\}$, runs in time polynomial in $1/\epsilon$ and in k and outputs, with high probability over its internal coin tosses, a set $S \subseteq \{0,1\}^k$ that contains all the strings $a \in \{0,1\}^k$ such that ℓ_a and f agree on at least a $1/2 + \epsilon$ fraction of inputs.*

Theorem 1 is proved by Goldreich and Levin [3]. The result can be seen as a list-decoding for the Hadamard code $H : \{0,1\}^k \to \{0,1\}^{2^k}$; remarkably, the algorithm runs in time polynomial in k, which is poly-logarithmic in the length of the given corrupted encoding.

Theorem 2 *There is a randomized algorithm KM that given in input an integer k and parameters $\epsilon, \delta > 0$, and given oracle access to a function $f : \{0,1\}^k \to \{0,1\}$, runs in time polynomial in $1/\epsilon$, in $1/\delta$, and in k and outputs a set $S \subseteq \{0,1\}^k$ and a value $g(a)$ for each $a \in S$.*

With high probability over the internal coin tosses of the algorithm,

1 *S contains all the strings $a \in \{0,1\}^k$ such that $|\hat{f}(a)| \geq \epsilon$, and*
2 *For every $a \in S$, $|\hat{f}(a) - g(a)| \leq \delta$.*

Theorem 2 is proved by Kushilevitz and Mansour [5]; it is an easy consequence of the Goldreich–Levin algorithm.

Applications

There are two key applications of the Goldreich–Levin algorithm: one is to cryptography and the other is to computational learning theory.

Application in Cryptography

In cryptography, a *one-way permutation* is a family of functions $\{p_n\}_{n \geq 1}$ such that: (i) for every n, $p_n : \{0,1\}^n \to \{0,1\}^n$ is bijective, (ii) there is a polynomial time algorithm that, given $x \in \{0,1\}^n$, computes $p_n(x)$, and (iii) for every polynomial time algorithm A and polynomial q, and for every sufficiently large n,

$$\mathbb{P}_{x \sim \{0,1\}^n}[A(p_n(x)) = x] \leq \frac{1}{q(n)} .$$

That is, even though computing $p_n(x)$ given x is doable in polynomial time, the task of computing x given $p_n(x)$ is intractable. A *hard core predicate* for a one-way permutation $\{p_n\}$ is a family of functions $\{B_n\}_{n \geq 1}$ such that: (i) for every n, $B_n : \{0,1\}^n \to \{0,1\}$, (ii) there is a polynomial time algorithm that, given $x \in \{0,1\}^n$, computes $B_n(x)$, and (iii) for every polynomial time algorithm A and polynomial q, and for every sufficiently large n,

$$\mathbb{P}_{x \sim \{0,1\}^n}[A(p_n(x)) = B_n(x)] \leq \frac{1}{2} + \frac{1}{q(n)} .$$

That is, even though computing $B_n(x)$ given x is doable in polynomial time, the task of computing $B_n(x)$ given $p_n(x)$ is intractable.

Goldreich and Levin [3] use their algorithm to show that every one-way permutation has a hard-core predicate, as stated in the next theorem.

Theorem 3 *Let $\{p_n\}$ be a one-way permutation; define $\{p'_n\}$ such that $p'_{2n}(x,y) := p_n(x), y$ and let $B_{2n}(x,y) := \sum_i x_i y_i \bmod 2$. (For odd indices, let $p'_{2n+1}(z,b) := p'_{2n}(z)$ and $B_{2n+1}(z,b) := B_{2n}(z)$.)*

Then $\{p'_n\}$ is a one-way permutation and $\{B_n\}$ is a hard-core predicate for $\{p'_n\}$.

This result is used in efficient constructions of pseudorandom generators, pseudorandom functions, and private-key encryption schemes based on one-way permutations. The interested reader is referred to Chapter 3 in Goldreich's monograph [2] for more details.

There are also related applications in computational complexity theory, especially in the study of average-case complexity. See [7] for an overview.

Application in Computational Learning Theory

Loosely speaking, in computational learning theory one is given an unknown function $f : \{0,1\}^k \to \{0,1\}$ and one wants to compute, via an efficient randomized algorithm, a representation of a function $g : \{0,1\}^k \to \{0,1\}$ that agrees with f on most inputs. In the *PAC learning* model, one has access to f only via randomly sampled pairs $(x, f(x))$; in the model of *learning with queries*, instead, one can evaluate f at points of one's choice. Kushilevitz and Mansour [5] suggest the following algorithm: using the algorithm of Theorem 2, find a set S of large coefficients and approximations $g(a)$ of the coefficients $\hat{f}(a)$ for $a \in S$. Then define the function $g(x) = \sum_{a \in S} g(a)\chi_a(x)$. If the error caused by the absence of the smaller coefficients and the imprecision in the larger coefficient is not too large, g and f will agree on most inputs. (A technical point is that g as defined above is not necessarily a boolean function, but it can be easily "rounded" to be boolean.) Kushilevitz and Mansour show that such an approach works well for the class of functions f for which $\sum_a |\hat{f}(a)|$ is bounded, and they observe that functions of small *decision tree complexity* fall into this class. In particular, they derive the following result.

Theorem 4 *There is a randomized algorithm that, given in input parameters k, m, ε and δ, and given oracle access to a function $f : \{0,1\}^k \to \{0,1\}$ of decision tree complexity at most m, runs in time polynomial in k, m, $1/\epsilon$ and $\log 1/\delta$ and, with probability at least $1 - \delta$ over its internal coin tosses, outputs a circuit computing a function $g : \{0,1\}^k \to \{0,1\}$ that agrees with f on at least a $1 - \epsilon$ fraction of inputs.*

Another application of the Kushilevitz–Mansour technique is due to Linial, Mansour, and Nisan [6].

Cross-References

► Decoding Reed–Solomon Codes

Recommended Reading

1. Elias, P.: List decoding for noisy channels. Technical Report 335, Research Laboratory of Electronics, MIT, Campridge, MA, USA (1957)
2. Goldreich, O.: The Foundations of Cryptography – Volume 1. Cambridge University Press, Campridge, UK (2001)
3. Goldreich, O., Levin, L.: A hard-core predicate for all one-way functions. In: Proceedings of the 21st ACM Symposium on Theory of Computing, pp. 25–32 Seattle, 14–17 May 1989
4. Hamming, R.: Error detecting and error correcting codes. Bell Syst. Tech. J. **29**, 147–160 (1950)
5. Kushilevitz, E., Mansour, Y.: Learning decision trees using the fourier spectrum. SIAM J. Comp. **22**(6), 1331–1348 (1993)
6. Linial, N., Mansour, Y., Nisan, N.: Constant depth circuits, fourier transform and learnability. J. ACM **40**(3), 607–620 (1993)
7. Trevisan, L.: Some applications of coding theory in computational complexity. Quaderni Matematica **13**, 347–424 (2004) arXiv:cs.CC/0409044

Learning with Malicious Noise

1993; Kearns, Li

PETER AUER
Institute for Computer Science, University of Leoben, Leoben, Austria

Problem Definition

This problem is concerned with PAC learning of concept classes when training examples are effected by malicious errors. The PAC (probably approximately correct) model of learning (also known as the distribution-free model of learning) was introduced by Valiant [11]. This model makes the idealized assumption that error-free training examples are generated from the same distribution which is then used to evaluate the learned hypothesis. In many environments, however, there is some chance that an erroneous example is given to the learning algorithm. The malicious noise model – again introduced by Valiant [12] – extends the PAC model by allowing example errors of any kind: it makes no assumptions on the nature of the errors that occur. In this sense the malicious noise model is a worst-case model of errors, in which errors may be generated by an adversary whose goal is to foil the learning algorithm. Kearns and Li [7,8] study the maximal malicious error rate such that learning is still possible. They also provide a canonical method to transform any standard learning algorithm into an algorithm which is robust against malicious noise.

Notations

Let X be a set of instances. The goal of a learning algorithm is to infer an unknown subset $C \subseteq X$ of instances which exhibit a certain property. Such subsets are called concepts. It is known to the learning algorithm that the correct concept C is from a concept class $\mathcal{C} \subseteq 2^X$, $C \in \mathcal{C}$. Let $C(x) = 1$ if $x \in C$ and $C(x) = 0$ if $x \notin C$. As input the learning algorithm receives an accuracy parameter $\varepsilon > 0$, a confidence parameter $\delta > 0$, and the malicious noise rate $\beta \geq 0$. The learning algorithm may request a sample of labeled instances $S = \langle (x_1, \ell_1), \ldots, (x_m, \ell_m) \rangle$, $x_i \in X$ and $\ell_i \in \{0,1\}$, and produces a hypothesis $H \subseteq X$. Let $\mathcal{D}$ be the unknown distribution of instances in X. Learning is

successful if H misclassifies an example with probability less than ε, $\mathrm{err}_{\mathcal{D}}(C,H) := \mathcal{D}\{x \in X : C(x) \neq H(x)\} < \varepsilon$. A learning algorithm is requested to be successful with probability $1-\delta$. The error of a hypothesis H in respect to a sample S of labeled instances is defined as $\mathrm{err}(S,H) := |\{(x,\ell) \in S : H(x) \neq \ell\}|/|S|$.

The VC-dimension $\mathrm{VC}(C)$ of a concept class C is the maximal number of instances $x_1,\ldots,x_d$ such that $\{(C(x_1),\ldots,C(x_d)) : C \in C\} = \{0,1\}^d$. The VC-dimension is a measure for the difficulty to learn concept class C [3].

To investigate the computational complexity of learning algorithms, sequences of concept classes with increasing complexity $(X_n, C_n)_n = \langle (X_1,C_1),(X_2,C_2),\ldots\rangle$ are considered. In this case the learning algorithm receives also a complexity parameter n as input.

Generation of Examples

In the malicious noise model the labeled instances (x_i,ℓ_i) are generated independently from each other by the following random process.

(a) Correct examples: with probability $1-\beta$ an instance x_i is drawn from distribution $\mathcal{D}$ and labeled by the correct concept C, $\ell_i = C(x_i)$.
(b) Noisy examples: with probability β an arbitrary example (x_i,ℓ_i) is generated, possibly by an adversary.

Problem 1 (Malicious Noise Learning of (X, C))
INPUT: *Reals $\varepsilon,\delta > 0$, $\beta \geq 0$.*
OUTPUT: *A hypothesis $H \subseteq X$.*
For any distribution $\mathcal{D}$ on X and any concept $C \in C$, the algorithm needs to produce with probability $1-\delta$ a hypothesis H such that $\mathrm{err}_{\mathcal{D}}(C,H) < \varepsilon$. The probability $1-\delta$ is taken in respect to the random sample $(x_1,\ell_1),\ldots,(x_m,\ell_m)$ requested by the algorithm. The examples (x_i,ℓ_i) are generated as defined above.

Problem 2 (Polynomial Malicious Noise Learning of $(X_n, C_n)_n$)
INPUT: *Reals $\varepsilon,\delta > 0$, $\beta \geq 0$, integer $n \geq 1$.*
OUTPUT: *A hypothesis $H \subseteq X_n$.*
For any distribution $\mathcal{D}$ on X_n and any concept $C \in C_n$, the algorithm needs to produce with probability $1-\delta$ a hypothesis H such that $\mathrm{err}_{\mathcal{D}}(C,H) < \varepsilon$. The computational complexity of the algorithm must be bounded by a polynomial in $1/\varepsilon$, $1/\delta$, and n.

Key Results

Theorem 1 ([8]) *If there are concepts $C_1, C_2 \in C$ and instances $x_1, x_2 \in X$ such that $C_1(x_1) = C_1(x_2)$ and $C_2(x_1) \neq C_2(x_2)$, then no algorithm learns C with malicious noise rate $\beta \geq \varepsilon/(1+\varepsilon)$.*

Theorem 2 *Let $\Delta > 0$ and $d = \mathrm{VC}(C)$. For a suitable constant κ, any algorithm which requests a sample S of $m \geq \kappa(\varepsilon d \log 1/(\Delta\delta))/\Delta^2$ labeled examples and returns a hypothesis $H \in C$ which minimizes $\mathrm{err}(S,H)$, learns the concept class C with malicious noise rate $\beta \leq \varepsilon/(1+\varepsilon) - \Delta$.*

Lower bounds on the number of examples necessary for learning with malicious noise were derived by Cesa-Bianchi et al. [5].

Theorem 3 ([5]) *Let $\Delta > 0$ and $d = \mathrm{VC}(C) \geq 3$. There is a constant κ, such that any algorithm which learns C with malicious noise rate $\beta = \varepsilon/(1+\varepsilon) - \Delta$ by requesting a sample and returning a hypothesis $H \in C$ which minimizes $\mathrm{err}(S,H)$, needs a sample of size at least $m \geq \kappa\varepsilon d/\Delta^2$.*

A general conversion of a learning algorithm for the noise-free model into an algorithm for the malicious noise model was given by Kearns and Li.

Theorem 4 ([8]) *Let $\mathcal{A}$ be a (polynomial-time) learning algorithm which learns concept classes C_n from $m(\varepsilon,\delta,n)$ noise-free examples, i. e. $\beta = 0$. Then $\mathcal{A}$ can be converted into a (polynomial-time) learning algorithm for C_n for any malicious noise rate $\beta \leq \log m(\varepsilon/8, 1/2, n)/m(\varepsilon/8, 1/2, n)$.*

The next theorem relates learning with malicious noise to a type of combinatorial optimization problems.

Theorem 5 ([8]) *Let $r \geq 1$ and $\alpha > 1$.*
1. *Let $\mathcal{A}$ be an algorithm which, for any sample S, returns a hypothesis $H \in C$ with $\mathrm{err}(S,H) \leq r \cdot \min_{C \in C} \mathrm{err}(S,C)$. Then $\mathcal{A}$ learns concept class C for any malicious noise rate $\beta \leq \varepsilon/(\alpha(1+\varepsilon)r)$ from a sufficiently large sample.*
2. *Let $\mathcal{A}$ be a polynomial-time learning algorithm for concept classes C_n which tolerates a malicious noise rate $\beta = \varepsilon/r$. Then $\mathcal{A}$ can be converted into a polynomial-time algorithm which for any sample S, with high probability returns a hypothesis $H \in C_n$ such that $\mathrm{err}(S,H) \leq \alpha r \cdot \min_{C \in C} \mathrm{err}(S,C)$.*

The computational hardness of several such related combinatorial optimization problems was shown by Ben-David, Eiron, and Long [2].

Applications

Several extensions of the learning model with malicious noise have been proposed, in particular the agnostic learning model [9] and the statistical query model [6]. Follow-

ing relations between these models and the malicious noise model have been established.

Theorem 6 ([9]) *If concept class C is polynomial-time learnable in the agnostic model, then C is polynomial-time learnable with any malicious noise rate $\beta \leq \varepsilon/2$.*

Theorem 7 ([6]) *If C is learnable from (relative error) statistical queries, then C is learnable with any malicious noise rate $\beta \leq \varepsilon/\log^p(1/\varepsilon)$ for a suitable large p independent of C.*

Another learning model related to the malicious noise model is learning with nasty noise [4]. In this model examples effected by malicious noise are not chosen at random with probability β, but an adversary might manipulate an arbitrary fraction of βm examples out of a given sample of size m. The malicious noise model was also considered in the context of on-line learning [1] and boosting [10].

Cross References

- ▶ Boosting Textual Compression
- ▶ PAC Learning
- ▶ Perceptron Algorithm
- ▶ Statistical Query Learning

Recommended Reading

1. Auer, P., Cesa-Bianchi, N.: On-line learning with malicious noise and the closure algorithm. Ann. Math. Artif. Intell. **23**, 83–99 (1998)
2. Ben-David, S., Eiron, N., Long, P.: On the difficulty of approximately maximizing agreements. J. CSS **66**, 496–514 (2003)
3. Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.: Learnability and the Vapnik-Chervonenkis dimension. J. ACM **36**, 929–965 (1989)
4. Bshouty, N., Eiron, N., Kushilevitz, E.: PAC learning with nasty noise. TCS **288**, 255–275 (2002)
5. Cesa-Bianchi, N., Dichterman, E., Fischer, P., Shamir, E., Simon, H.U.: Sample-efficient strategies for learning in the presence of noise. J. ACM **46**, 684–719 (1999)
6. Aslam, J.A., Decatur, S.E.: Specification and simulation of statistical query algorithms for efficiency and noise tolerance. J. CSS **56**, 191–208 (1998)
7. Kearns, M., Li, M.: Learning in the presence of malicious errors. In: Proc. 20th ACM Symp. Theory of Computing, pp. 267–280, Chicago, 2–4 May 1988
8. Kearns, M., Li, M.: Learning in the presence of malicious errors. SIAM J. Comput. **22**, 807–837 (1993)
9. Kearns, M., Schapire, R., Sellie, L.: Toward efficient agnostic learning. Mach. Learn. **17**, 115–141 (1994)
10. Servedio, R.A.: Smooth boosting and learning with malicious noise. JMLR **4**, 633–648 (2003)
11. Valiant, L.: A theory of the learnable. C. ACM **27**, 1134–1142 (1984)
12. Valiant, L.: Learning disjunctions of conjunctions. In: Proc. 9th Int. Joint Conference on Artificial Intelligence, pp. 560–566, Los Angeles, August 1985

Learning Significant Fourier Coefficients over Finite Abelian Groups

2003; Akavia, Goldwasser, Safra

Adi Akavia
Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, USA

Keywords and Synonyms

Learning heavy fourier coefficients; Finding heavy fourier coefficients

Problem Definition

Fourier transform is among the most widely used tools in computer science. Computing the Fourier transform of a signal of length N may be done in time $\Theta(N \log N)$ using the Fast Fourier Transform (FFT) algorithm. This time bound clearly cannot be improved below $\Theta(N)$, because the output itself is of length N. Nonetheless, it turns out that in many applications it suffices to find only the *significant Fourier coefficients*, i. e., Fourier coefficients occupying, say, at least 1% of the energy of the signal. This motivates the problem discussed in this entry: the problem of efficiently finding and approximating the significant Fourier coefficients of a given signal (SFT, in short). A naive solution for SFT is to first compute the entire Fourier transform of the given signal and then to output only the significant Fourier coefficients; thus yielding no complexity improvement over algorithms computing the entire Fourier transform. In contrast, SFT can be solved far more efficiently in running time $\widetilde{\Theta}(\log N)$ and while reading at most $\widetilde{\Theta}(\log N)$ out of the N signal's entries [2]. This fast algorithm for SFT opens the way to applications taken from diverse areas including computational learning, error correcting codes, cryptography, and algorithms.

It is now possible to formally define the SFT problem, restricting our attention to discrete signals. Use functional notation where a signal is a function $f: G \to \mathbb{C}$ over a finite Abelian group G, its energy is $\|f\|_2^2 \stackrel{\text{def}}{=} 1/|G| \sum_{x \in G} |f(x)|^2$, and its maximal amplitude is $\|f\|_\infty \stackrel{\text{def}}{=} \max\{|f(x)| \; | x \in G\}$.[1] For ease of presentation

[1] For readers more accustomed to vector notation, the authors remark that there is a simple correspondence between vector and functional notation. For example, a one-dimensional signal $(v_1, \ldots, v_N) \in \mathbb{C}^{\mathbb{N}}$ corresponds to the function $f: \mathbb{Z}_N \to \mathbb{C}$ defined by $f(i) = v_i$ for all $i = 1, \ldots, N$. Likewise, a two-dimensional signal $M \in \mathbb{C}^{N_1 \times N_2}$ corresponds to the function $f: \mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2} \to \mathbb{C}$ defined by $f(i, j) = M_{ij}$ for all $i = 1, \ldots, N_1$ and $j = 1, \ldots, N_2$.

assume without loss of generality that $G = \mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2} \times \cdots \times \mathbb{Z}_{N_k}$ for $N_1, \ldots, N_k \in \mathbb{Z}^+$ (i. e., positive integers), and for $\mathbb{Z}_N$ is the additive group of integers modulo N.

The *Fourier transform* of f is the function $\widehat{f}\colon G \to \mathbb{C}$ defined for each $\alpha = (\alpha_1, \ldots, \alpha_k) \in G$ by

$$\widehat{f}(\alpha) \stackrel{\text{def}}{=} \frac{1}{|G|} \sum_{(x_1,\ldots,x_k)\in G} \left[f(x_1, \ldots, x_k) \prod_{j=1}^{k} \omega_{N_j}^{\alpha_j x_j} \right],$$

where $\omega_{N_j} = \exp(i2\pi/N_j)$ is a primitive root of unity of order N_j. For any $\alpha \in G$, $val_\alpha \in \mathbb{C}$ and $\tau, \varepsilon \in [0,1]$, say that α is a *τ-significant Fourier coefficient* iff $|\widehat{f}(\alpha)|^2 \geq \tau \|f\|_2^2$, and say that val_α is an *ε-approximation for* $\widehat{f}(\alpha)$ iff $|val_\alpha - \widehat{f}(\alpha)| < \varepsilon$.

Problem 1 (SFT)

INPUT: *Integers $N_1, \ldots, N_k \geq 2$ specifying the group $G = \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_k}$, a threshold $\tau \in (0,1)$, an approximation parameter $\varepsilon \in (0,1)$, and oracle access[2] to $f\colon G \to \mathbb{C}$.*

OUTPUT: *A list of all τ-significant Fourier coefficients of f along with ε-approximations for them.*

Key Results

The key result of this entry is an algorithm solving the SFT problem which is much faster than algorithms for computing the entire Fourier transform. For example, for f a Boolean function over $\mathbb{Z}_N$, the running time of this algorithm is $\log N \cdot poly(\log\log N, 1/\tau, 1/\varepsilon)$, in contrast to the $\Theta(N \log N)$ running time of the FFT algorithm. This algorithm is named the SFT algorithm.

Theorem 1 (SFT algorithm [2]) *There is an algorithm solving the SFT problem with running time $\log|G| \cdot poly(\log\log|G|, \|f\|_\infty/\|f\|_2, 1/\tau, 1/\varepsilon)$ for $|G| = \prod_{j=1}^{k} N_j$ the cardinality of G.*

Remarks

1. The above result extends to functions f over any finite Abelian group G, as long as the algorithm is given a description of G by its generators and their orders [2].
2. The SFT algorithm reads at most $\log|G| \cdot poly(\log\log|G|, \|f\|_\infty/\|f\|_2, 1/\tau, 1/\varepsilon)$ out of the $|G|$ values of the signal.
3. The SFT algorithm is non adaptive, that is, oracle queries to f are independent of the algorithm's progress.
4. The SFT algorithm is a probabilistic algorithm having a small error probability, where probability is taken over the internal coin tosses of the algorithm. The error probability can be made arbitrarily small by standard amplification techniques.

The SFT algorithm follows a line of works solving the SFT problem for restricted function classes. Goldreich and Levin [9] gave an algorithm for Boolean functions over the group $\mathbb{Z}_2^k = \{0,1\}^k$. The running time of their algorithm is polynomial in $k, 1/\tau$ and $1/\varepsilon$. Mansour [10] gave an algorithm for complex functions over groups $G = \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_k}$ provided that $N_1, \ldots, N_k$ are powers of two. The running time of his algorithm is polynomial in $\log|G|, \log(\max_{\alpha\in G}|\widehat{f}(\alpha)|), 1/\tau$ and $1/\varepsilon$. Gilbert et al. [6] gave an algorithm for complex functions over the group $\mathbb{Z}_N$ for any positive integer N. The running time of their algorithm is polynomial in $\log N$, $\log(\max_{x\in\mathbb{Z}_N} f(x)/\min_{x\in\mathbb{Z}_N} f(x)), 1/\tau$ and $1/\varepsilon$. Akavia et al. [2] gave an algorithm for complex functions over any finite Abelian group. The latter [2] improves on [6] even when restricted to functions over $\mathbb{Z}_N$ in achieving $\log N \cdot poly(\log\log N)$ rather than $poly(\log N)$ dependency on N. Subsequent works [7] improved the dependency of [6] on τ and ε.

Applications

Next, the paper surveys applications of the SFT algorithm [2] in the areas of computational learning theory, coding theory, cryptography, and algorithms.

Applications in Computational Learning Theory

A common task in computational learning is to find a hypothesis h approximating a function f, when given only samples of the function f. Samples may be given in a variety of forms, e. g., via oracle access to f. We consider the following variant of this learning problem: f and h are complex functions over a finite Abelian group $G = \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_k}$, the goal is to find h such that $\|f - h\|_2^2 \leq \gamma \|f\|_2^2$ for $\gamma > 0$ an approximation parameter, and samples of f are given via oracle access.

A straightforward application of the SFT algorithm gives an efficient solution to the above learning problem, provided that there is a small set $\Gamma \subseteq G$ s.t. $\sum_{\alpha\in\Gamma} |\widehat{f}(\alpha)|^2 > (1 - \gamma/3)\|f\|_2^2$. The learning algorithm operates as follows. It first runs the SFT algorithm to find all $\alpha = (\alpha_1, \ldots, \alpha_k) \in G$ that are $\gamma/|\Gamma|$-significant Fourier coefficients of f along with their $\gamma/|\Gamma|\|f\|_\infty$-

[2] Say that an algorithm is given *oracle access* to a function f over G, if it can request and receive the value $f(x)$ for any $x \in G$ in unit time.

approximations val_α, and then returns the hypothesis

$$h(x_1,\ldots,x_k) \stackrel{\text{def}}{=} \sum_{\alpha \text{ is } \gamma/|\Gamma|-\text{significant}} val_\alpha \cdot \prod_{j=1}^{k} \omega_{N_j}^{\alpha_j x_j}.$$

This hypothesis h satisfies that $\|f-h\|_2^2 \le \gamma\|f\|_2^2$. The running time of this learning algorithm and the number of oracle queries it makes is polynomially bounded by $\log|G|, \|f\|_\infty/\|f\|_2, |\Gamma|\|f\|_\infty/\gamma$.

Theorem 2 *Let $f: G \to \mathbb{C}$ be a function over $G = \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_k}$, and $\gamma > 0$ an approximation parameter. Denote $t = \min\{|\Gamma| : \Gamma \subseteq G \text{ s.t. } \sum_{\alpha\in\Gamma} |\widehat{f}(\alpha)|^2 > (1-\gamma/3)\|f\|_2^2\}$. There is an algorithm that given $N_1,\ldots,N_k$, γ, and oracle access to f, outputs a (short) description of $h: G \to \mathbb{C}$ s.t. $\|f-h\|_2^2 < \gamma\|f\|_2^2$. The running time of this algorithm is $\log|G| \cdot poly(\log\log|G|, \|f\|_\infty/\|f\|_2, t\|f\|_\infty/\gamma)$.*

More examples of function classes that can be efficiently learned using our SFT algorithm are given in [3].

Applications in Coding Theory

Error correcting codes encode messages in a way that allows *decoding*, that is, recovery of the original message, even in the presence of noise. When the noise is very high, unique decoding may be infeasible, nevertheless it may still be possible to *list decode*, that is, to find a short list of messages containing the original message. Codes equipped with an efficient list decoding algorithm have found many applications (see [11] for a survey).

Formally, a binary code is a subset $C \subseteq \{0,1\}^*$ of *codewords* each encoding some message. Denote by $C_{N,x} \in \{0,1\}^N$ a codeword of length N encoding a message x. The *normalized Hamming distance* between a codeword $C_{N,x}$ and a received word $w \in \{0,1\}^N$ is $\Delta(C_{N,x}, w) \stackrel{\text{def}}{=} 1/N|\{i \in \mathbb{Z}_N : C_{N,x}(i) \neq w(i)\}|$ where $C_{N,x}(i)$ and $w(i)$ are the ith bits of $C_{N,x}$ and w, respectively. Given $w \in \{0,1\}^N$ and a noise parameter $\eta > 0$, the *list decoding* task is to find a list of all messages x such that $\Delta(C_{N,x}, w) < \eta$. The received word w may be given explicitly or implicitly; we focus on the latter where oracle access to w is given. Goldreich and Levin [9] give a list decoding algorithm for Hadamard codes, using in a crucial way their algorithm solving the SFT problem for functions over the Boolean cube.

The SFT algorithm for functions over ZZ_N is a key component in a list decoding algorithm given by Akavia et al. [2]. This list decoding algorithm is applicable to a large class of codes. For example, it is applicable to the code $C^{msb} = \{C_{N,x}: \mathbb{Z}_N \to \{0,1\}\}_{x\in\mathbb{Z}_N^*, N\in\mathbb{Z}^+}$ whose codewords are $C_{N,x}(j) = msb_N(j\cdot x \mod N)$ for $msb_N(y) = 1$ iff $y \ge N/2$ and $msb_N(y) = 0$ otherwise. More generally, this list decoding algorithm is applicable to any *Multiplication code* $C^{\mathcal{P}}$ for $\mathcal{P}$ a family of *balanced* and *well concentrated* functions, as defined below. The running time of this list decoding algorithm is polynomial in $\log N$ and $1/(1-2\eta)$, as long as $\eta < \frac{1}{2}$.

Abstractly, the list decoding algorithm of [2] is applicable to any code that is "balanced," "(well) concentrated," and "recoverable," as defined next (and those Fourier coefficients have small greatest common divisor (GCD) with N). A code is *balanced* if $\Pr_{j\in\mathbb{Z}_N}[C_{N,x}(j)=0] = \Pr_{j\in\mathbb{Z}_N}[C_{N,x}(j)=1]$ for every codeword $C_{N,x}$. A code is *(well) concentrated* if its codewords can be approximated by a small number of significant coefficients in their Fourier representation (and those Fourier coefficients have small greatest common divisor (GCD) with N). A code is *recoverable* if there is an efficient algorithm mapping each Fourier coefficient α to a short list of codewords for which α is a significant Fourier coefficient. The key property of concentrated codes is that received words w share a significant Fourier coefficient with all close codewords $C_{N,x}$. The high level structure of the list decoding algorithm of [2] is therefore as follows. First it runs the SFT algorithm to find all significant Fourier coefficients α of the received word w. Second for each such α, it runs the recovery algorithm to find all codewords $C_{N,x}$ for which α is significant. Finally, it outputs all those codewords $C_{N,x}$.

Definition 1 (Multiplication codes [2])
Let $\mathcal{P} = \{P_N: \mathbb{Z}_N \to \{0,1\}\}_{N\in\mathbb{Z}^+}$ be a family of functions. Say that $C^{\mathcal{P}} = \{C_{N,x}: \mathbb{Z}_N \to \{0,1\}\}_{x\in\mathbb{Z}_N^*, N\in\mathbb{Z}^+}$ is a *multiplication code for* $\mathcal{P}$ if for every $N \in \mathbb{Z}^+$ and $x \in \mathbb{Z}_N^*$, the encoding $C_{N,x}: \mathbb{Z}_N \to \{0,1\}$ of x is defined by

$$C_{N,x}(j) = P(j\cdot x \bmod N).$$

Definition 2 (Well concentrated [2]) Let $\mathcal{P} = \{P_N: \mathbb{Z}_N \to \mathbb{C}\}_{N\in\mathbb{Z}^+}$ be a family of functions. Say that $\mathcal{P}$ is *well concentrated* if $\forall N \in \mathbb{Z}^+, \gamma > 0, \exists \Gamma \subseteq \mathbb{Z}_N$ s.t. (i) $|\Gamma| \le poly(\log N/\gamma)$, (ii) $\sum_{\alpha\in\Gamma} |\widehat{P_N}(\alpha)|^2 \ge (1-\gamma)\|P_N\|_2^2$, and (iii) for all $\alpha \in \Gamma$, $gcd(\alpha, N) \le poly(\log N/\gamma)$ (where $gcd(\alpha, N)$ is the greatest common divisor of α and N).

Theorem 3 (List decoding [2]) *Let $\mathcal{P} = \{P_N: \mathbb{Z}_N \to \{0,1\}\}_{N\in\mathbb{Z}^+}$ be a family of efficiently computable[3], well concentrated, and balanced functions. Let $C^{\mathcal{P}} =$*

[3] $\mathcal{P} = \{P_N: \mathbb{Z}_N \to \{0,1\}\}_{N\in\mathbb{Z}^+}$ is a family of *efficiently computable* functions if there is an algorithm that given any $N \in \mathbb{Z}^+$ and $x \in \mathbb{Z}_N$ outputs $P_N(x)$ in time $poly(\log N)$.

$\{C_{N,x}: \mathbb{Z}_N \to \{0,1\}\}_{x\in\mathbb{Z}_N^*, N\in\mathbb{Z}^+}$ *be the multiplication code for* $\mathcal{P}$. *Then there is an algorithm that, given* $N \in \mathbb{Z}_N^+$, $\eta < \frac{1}{2}$ *and oracle access to* $w: \mathbb{Z}_N \to \{0,1\}$, *outputs all* $x \in \mathbb{Z}_N^*$ *for which* $\Delta(C_{N,x}, w) < \eta$. *The running time of this algorithm is polynomial in* $\log N$ *and* $1/(1-2\eta)$.

Remarks

1. The requirement that $\mathcal{P}$ is a family of *efficiently computable* functions can be relaxed. It suffices to require that the list decoding algorithm receives or computes a set $\Gamma \subseteq \mathbb{Z}_N$ with properties as specified in Definition 2.
2. The requirement that $\mathcal{P}$ is a family of *balanced* functions can be relaxed. Denote $bias(\mathcal{P}) = \min_{b\in\{0,1\}} \inf_{N\in\mathbb{Z}^+} \Pr_{j\in\mathbb{Z}_N}[P_N(j) = b]$. Then the list decoding algorithm of [2] is applicable to $C^{\mathcal{P}}$ even when $bias(\mathcal{P}) \neq \frac{1}{2}$, as long as $\eta < bias(\mathcal{P})$.

Applications in Cryptography

Hard-core predicates for one-way functions are a fundamental cryptographic primitive, which is central for many cryptographic applications such as pseudo-random number generators, semantic secure encryption, and cryptographic protocols. Informally speaking, a Boolean predicate P is a *hard-core predicate* for a function f if $P(x)$ is easy to compute when given x, but hard to guess with a non-negligible advantage beyond 50% when given only $f(x)$. The notion of hardcore predicates was introduced by Blum and Micali [2]. Goldreich and Levin [9] showed a randomized hardcore predicate for any one-way function, using in a crucial way their algorithm solving the SFT problem for functions over the Boolean cube.

Akavia et al. [2] introduce a unifying framework for proving that a predicate P is hard-core for a one-way function f. Applying their framework they prove for a wide class of predicates—*segment predicates*—that they are hard-core predicates for various well-known candidate one-way functions. Thus showing new hard-core predicates for well-known one-way function candidates as well as reproving old results in an entirely different way.

Elaborating on the above, a *segment predicate* is any assignment of Boolean values to an arbitrary partition of $\mathbb{Z}_N$ into $poly(\log N)$ segments, or dilations of such an assignment. Akavia et al. [2] prove that any segment predicate is hard-core for any one-way function f defined over $\mathbb{Z}_N$ for which, for a non-negligible fraction of the x's in $\mathbb{Z}_N$, given $f(x)$ and y, one can efficiently compute $f(xy)$ (where xy is multiplication in $\mathbb{Z}_N$). This includes the following functions: the *exponentiation* function $EXP_{p,g}: \mathbb{Z}_p \to \mathbb{Z}_p^*$ defined by $EXP_{p,g}(x) = g^x \bmod p$ for each prime p and a generator g of the group $\mathbb{Z}_p^*$; the *RSA* function $RSA: \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ defined by $RSA(x) = e^x \bmod N$ for each $N = pq$ a product of two primes p, q, and e co-prime to N; the *Rabin* function $Rabin: \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ defined by $Rabin(x) = x^2 \bmod N$ for each $N = pq$ a product of two primes p, q; and the *elliptic curve log* function defined by $ECL_{a,b,p,Q} = xQ$ for each elliptic curve $E_{a,b,p}(Z_p)$ and Q a point of high order on the curve.

The SFT algorithm is a central tool in the framework of [2]: Akavia et al. take a list decoding methodology, where computing a hard-core predicate corresponds to computing an entry in some error correcting code, predicting a predicate corresponds to access to an entry in a corrupted codeword, and the task of inverting a one-way function corresponds to the task of list decoding a corrupted codeword. The codes emerging in [2] are multiplication codes (see Definition 1 above), which are list decoded using the SFT algorithm.

Definition 3 (Segment predicates [2]) Let $\mathcal{P} = \{P_N: \mathbb{Z}_N \to \{0,1\}\}_{N\in\mathbb{Z}^+}$ be a family of predicates that are non-negligibly far from constant[4].

- It can be sayed that P_N is a basic t-segment predicate *if* $P_N(x+1) \neq P_N(x)$ *for at most* t x*'s in* $\mathbb{Z}_N$.
- It can be sayed that P_N is a t-segment predicate *if there exist a basic* t*-segment predicate* P' *and* $a \in \mathbb{Z}_N$ *which is co-prime to* N *s.t.* $\forall x \in \mathbb{Z}_N, P_N(x) = P'(x/a)$.
- It can be sayed that $\mathcal{P}$ is a family of segment predicates *if* $\forall N \in \mathbb{Z}^+$, P_N *is a* $t(N)$*-segment predicate for* $t(N) \leq poly(\log N)$.

Theorem 4 (Hardcore predicates [2]) *Let* $\mathcal{P}$ *be a family of segment predicates. Then,* $\mathcal{P}$ *is hard-core for RSA, Rabin, EXP, ECL, under the assumption that these are one-way functions.*

Application in Algorithms

Our modern times are characterized by information explosion incurring a need for faster and faster algorithms. Even algorithms classically regarded as efficient—such as the FFT algorithm with its $\Theta(N \log N)$ complexity—are often too slow for data-intensive applications, and linear or even sub-linear algorithms are imperative. Despite the vast variety of fields and applications where algorithmic challenges arise, some basic algorithmic building blocks emerge in many of the existing algorithmic solutions. Accelerating such building blocks can therefore accelerate

[4] A family of functions $\mathcal{P} = \{P_N: \mathbb{Z}_N \to \{0,1\}\}_{N\in\mathbb{Z}^+}$ is *non-negligibly far from constant* if $\forall N \in \mathbb{Z}^+$ and $b \in \{0,1\}$, $\Pr_{j\in\mathbb{Z}_N}[P_N(j) = b] \leq 1 - poly(1/\log N)$.

many existing algorithms. One of these recurring building blocks is the Fast Fourier Transform (FFT) algorithm. The SFT algorithm offers a great efficiency improvement over the FFT algorithm for applications where it suffices to deal only with the significant Fourier coefficients. In such applications, replacing the FFT building block with the SFT algorithm accelerates the $\Theta(N \log N)$ complexity in each application of the FFT algorithm to $poly(\log N)$ complexity [1]. Lossy compression is an example of such an application [1,5,8]. To elaborate, central component in several transform compression methods (e. g., JPEG) is to first apply Fourier (or Cosine) transform to the signal, and then discard many of its coefficients. To accelerate such algorithms —instead of computing the entire Fourier (or Cosine) transform—the SFT algorithm can be used to directly approximate only the significant Fourier coefficients. Such an accelerated algorithm achieves compression guarantee as good as the original algorithm (and possibly better), but with running time improved to $poly(\log N)$ in place of the former $\Theta(N \log N)$.

Cross References

- ▶ Abelian Hidden Subgroup Problem
- ▶ Learning Constant-Depth Circuits
- ▶ Learning DNF Formulas
- ▶ Learning Heavy Fourier Coefficients of Boolean Functions
- ▶ Learning with Malicious Noise
- ▶ List Decoding near Capacity: Folded RS Codes
- ▶ PAC Learning
- ▶ Statistical Query Learning

Recommended Reading

1. Akavia, A., Goldwasser, S.: Manuscript submitted as an NSF grant, awarded (2005) CCF-0514167
2. Akavia, A., Goldwasser, S., Safra, S.: Proving hard-core predicates using list decoding. In: Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS'03), pp. 146–157. IEEE Computer Society (2003)
3. Atici, A., Servedio, R.A.: Learning unions of $\omega(1)$-dimensional rectangles. In: ALT, pp. 32–47 (2006)
4. Blum, M., Micali, S.: How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. SIAM J. Comput. **4**(13), 850–864 (1984)
5. Cormode, G., Muthukrishnan, S.: Combinatorial algorithms for compressed sensing. In: Structural Information and Communication Complexity, 13th International Colloquium, SIROCCO (2006), Chester, UK, July 2–5, 2006 pp. 280–294
6. Gilbert, A.C., Guha, S., Indyk, P., Muthukrishnan, S., Strauss, M.: Near-optimal sparse fourier representations via sampling. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, pp. 152–161. ACM Press (2002)
7. Gilbert, A.C., Muthukrishnan, S., Strauss, M.J.: Improved time bounds for near-optimal sparse fourier representation via sampling. In: Proceedings of SPIE Wavelets XI, San Diego, CA 2005 (2005)
8. Gilbert, A.C., Strauss, M.J., Tropp, J.A., Vershynin, R.: One sketch for all: Fast algorithms for compressed sensing. In: 39th ACM Symposium on Theory of Computing (STOC'07)
9. Goldreich, O., Levin, L.: A hard-core predicate for all one-way functions. In: 27th ACM Symposium on Theory of Computing (STOC'89) (1989)
10. Mansour, Y.: Randomized interpolation and approximation of sparse polynomials. SIAM J. Comput. **24**, 357–368 (1995)
11. Sudan, M.: List decoding: algorithms and applications. SIGACT News **31**, 16–27 (2000)

LEDA: a Library of Efficient Algorithms

1995; Mehlhorn, Näher

Christos Zaroliagis
Department of Computer Engineering and Informatics, University of Patras, Patras, Greece

Keywords and Synonyms

LEDA platform for combinatorial and geometric computing

Problem Definition

In the last forty years, there has been a tremendous progress in the field of computer algorithms, especially within the core area known as *combinatorial algorithms*. Combinatorial algorithms deal with objects such as lists, stacks, queues, sequences, dictionaries, trees, graphs, paths, points, segments, lines, convex hulls, etc, and constitute the basis for several application areas including network optimization, scheduling, transport optimization, CAD, VLSI design, and graphics. For over thirty years, asymptotic analysis has been the main model for designing and assessing the efficiency of combinatorial algorithms, leading to major algorithmic advances.

Despite so many breakthroughs, however, very little had been done (at least until 15 years ago) about the practical utility and assessment of this wealth of theoretical work. The main reason for this lack was the absence of a standard *algorithm library*, that is, of a software library that contains a systematic collection of robust and efficient implementations of algorithms and data structures, upon which other algorithms and data structures can be easily built.

The lack of an algorithm library limits severely the great impact which combinatorial algorithms can have.

The continuous re-implementation of basic algorithms and data structures slows down progress and typically discourages people to make the (additional) effort to use an efficient solution, especially if such a solution cannot be re-used. This makes the migration of scientific discoveries into practice a very slow process.

The major difficulty in building a library of combinatorial algorithms stems from the fact that such algorithms are based on complex data types, which are typically not encountered in programming languages (i. e., they are not built-in types). This is in sharp contrast with other computing areas such as statistics, numerical analysis, and linear programming.

Key Results

The currently most successful algorithm library is LEDA (Library for Efficient Data types and Algorithms) [4,5]. It contains a very large collection of advanced data structures and algorithms for combinatorial and geometric computing. The development of LEDA started in the early 1990s, it reached a very mature state in the late 1990s, and it continues to grow. LEDA has been written in C++ and has benefited considerably from the object-oriented paradigm.

Four major goals have been set in the design of LEDA.

1. *Ease of use*: LEDA provides a sizable collection of data types and algorithms in a form that they can be readily used by non-experts. It gives a precise and readable specification for each data type and algorithm, which is short, general and abstract (to hide the details of implementation). Most data types in LEDA are parameterized (e. g., the dictionary data type works for arbitrary key and information type). To access the objects of a data structure by position, LEDA has invented the *item concept* that casts positions into an abstract form.
2. *Extensibility*: LEDA is easily extensible by means of parametric polymorphism and can be used as a platform for further software development. Advanced data types are built on top of basic ones, which in turn rest on a uniform conceptual framework and solid implementation principles. The main mechanism to extend LEDA is through the so-called LEDA extension packages (LEPs). A LEP extends LEDA into a particular application domain and/or area of algorithms that is not covered by the core system. Currently, there are 15 such LEPs; for details see [1].
3. *Correctness*: In LEDA, programs should give sufficient justification (proof) for their answers to allow the user of a program to easily assess its correctness. Many algorithms in LEDA are accompanied by *program checkers*. A program checker C for a program P is a (typically very simple) program that takes as input the input of P, the output of P, and perhaps additional information provided by P, and verifies that the answer of P in indeed the correct one.
4. *Efficiency*: The implementations in LEDA are usually based on the asymptotically most efficient algorithms and data structures that are known for a problem. Quite often, these implementations have been fine-tuned and enhanced with heuristics that considerably improve running times. This makes LEDA not only the most comprehensive platform for combinatorial and geometric computing, but also a library that contains the currently fastest implementations.

Since 1995, LEDA is maintained by the Algorithmic Solutions Software GmbH [1] which is responsible for its distribution in academia and industry.

Other efforts for algorithm libraries include the Standard Template Library (STL) [7], the Boost Graph Library [2,6], and the Computational Geometry Algorithms Library (CGAL) [3].

STL [7] (introduced in 1994) is a library of interchangeable components for solving many fundamental problems on sequences of elements, which has been adopted into the C++ standard. It contributed the *iterator concept* which provides an interface between *containers* (an object that stores other objects) and algorithms. Each algorithm in STL is a function template parameterized by the types of iterators upon which it operates. Any iterator that satisfies a minimum set of requirements can be used regardless of the data structure accessed by the iterator. The systematic approach used in STL to build abstractions and interchangeable components is called *generic programming*.

The Boost Graph Library [2,6] is a C++ graph library that applies the notions of generic programming to the construction of graph algorithms. Each graph algorithm is written not in terms of a specific data structure, but instead in terms of a graph abstraction that can be easily implemented by many different data structures. This offers the programmer the flexibility to use graph algorithms in a wide variety of applications. The first release of the library became available in September 2000.

The Computational Geometry Algorithms Library [3] is another C++ library that focuses on geometric computing only. Its main goal is to provide easy access to efficient and reliable geometric algorithms to users in industry and academia. The CGAL library started in 1996 and the first release was in April 1998.

Among all libraries mentioned above LEDA is by far the best (both in quality and efficiency of implementations) regarding combinatorial computing. It is worth

mentioning that the late versions of LEDA have also incorporated the iterator concept of STL.

Finally, a notable effort concerns the Stony Brook Algorithm Repository [8]. This is not an algorithm library, but a comprehensive collection of algorithm implementations for over seventy problems in combinatorial computing, started in 2001. The repository features implementations coded in different programming languages, including C, C++, Java, Fortran, ADA, Lisp, Mathematic, and Pascal.

Applications

An algorithm library for combinatorial and geometric computing has a wealth of applications in a wide variety of areas, including: network optimization, scheduling, transport optimization and control, VLSI design, computer graphics, scientific visualization, computer aided design and modeling, geographic information systems, text and string processing, text compression, cryptography, molecular biology, medical imaging, robotics and motion planning, and mesh partition and generation.

Open Problems

Algorithm libraries usually do not provide an interactive environment for developing and experimenting with algorithms. An important research direction is to add an interactive environment into algorithm libraries that would facilitate the development, debugging, visualization, and testing of algorithms.

Experimental Results

The are numerous experimental studies based on LEDA, STL, Boost, and CGAL, most of which can be found in the world-wide web. Also, the web sites of some of the libraries contain pointers to experimental work.

URL to Code

The afore mentioned algorithm libraries can be downloaded from their corresponding web sites, the details of which are given in the bibliography (Recommended Reading).

Cross References

- ► Engineering Algorithms for Large Network Applications
- ► Experimental Methods for Algorithm Analysis
- ► Implementation Challenge for Shortest Paths
- ► Shortest Paths Approaches for Timetable Information
- ► TSP-Based Curve Reconstruction

Recommended Reading

1. Algorithmic Solutions Software GmbH, http://www.algorithmic-solutions.com/. Accessed February 2008
2. Boost C++ Libraries, http://www.boost.org/. Accessed February 2008
3. CGAL: Computational Geometry Algorithms Library, http://www.cgal.org/. Accessed February 2008
4. Mehlhorn, K., Näher, S.: LEDA: A Platform for Combinatorial and Geometric Computing. Commun. ACM. **38**(1), 96–102 (1995)
5. Mehlhorn, K., Näher, S.. LEDA: A Platform for Combinatorial and Geometric Computing. Cambridge University Press, Boston (1999)
6. Siek, J., Lee, L.Q., Lumsdaine, A.: The Boost Graph Library. Addison-Wesley, Cambridge (2002)
7. Stepanov, A., Lee, M.: The Standard Template Library. In: Technical Report X3J16/94–0095, WG21/N0482, ISO Programming Language C++ Project. Hewlett-Packard, Palo Alto CA (1994)
8. The Stony Brook Algorithm Repository, http://www.cs.sunysb.edu/~algorith/. Accessed February 2008

Leontief Economy Equilibrium

2005; Codenotti, Saberi, Varadarajan, Ye
2005; Ye

Yin-Yu Ye
Department of Management Science and Engineering, Stanford University, Stanford, CA, USA

Keywords and Synonyms

Exchange market equilibrium with the leontief utility

Problem Definition

The Arrow–Debreu exchange market equilibrium problem was first formulated by Léon Walras in 1874 [7]. In this problem everyone in a population of m traders has an initial endowment of a divisible goods and a utility function for consuming all goods – their own and others'. Every trader sells the entire initial endowment and then uses the revenue to buy a bundle of goods such that his or her utility function is maximized. Walras asked whether prices could be set for everyone's goods such that this is possible. An answer was given by Arrow and Debreu in 1954 [1] who showed that, under mild conditions, such equilibrium would exist if the utility functions were concave. In general, it is unknown whether or not an equilibrium can be computed efficiently, see, e. g., ► General Equilibrium.

Consider a special class of Arrow–Debreu's problems, where each of the n traders has exactly one unit of a divisible and distinctive good for trade, and let trader i, $i = 1, \ldots, n$, bring good i, which class of problems is called the *pairing class*. For given prices p_j on good j, consumer

i's maximization problem is

$$\begin{array}{ll} \text{maximize} & u_i(x_{i1}, \ldots, x_{in}) \\ \text{subject to} & \sum_j p_j x_{ij} \le p_i, \\ & x_{ij} \ge 0, \quad \forall j\,. \end{array} \tag{1}$$

Let x_i^* denote a maximal solution vector of (1). Then, vector p is called the Arrow–Debreu price equilibrium if there exists an x_i^* for consumer i, $i = 1, \ldots, n$, such that

$$\sum_i x_i^* = e$$

where e is the vector of all ones representing available goods on the exchange market.

The Leontief Economy Equilibrium problem is the Arrow–Debreu Equilibrium problem when the utility functions are in the Leontief form:

$$u_i(x_i) = \min_{j:\, h_{ij}>0} \left\{ \frac{x_{ij}}{h_{ij}} \right\},$$

where the Leontief coefficient matrix is given by

$$H = \begin{pmatrix} h_{11} & h_{12} & \ldots & h_{1n} \\ h_{21} & h_{22} & \ldots & h_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ h_{n1} & h_{n2} & \ldots & h_{nn} \end{pmatrix}. \tag{2}$$

Here, one may assume that

Assumption 1 *H has no all-zero row, that is, every trader likes at least one good.*

Key Results

Let u_i be the equilibrium utility value of consumer i and p_i be the equilibrium price for good i, $i = 1, \ldots, n$. Also, let U and P are diagonal matrices whose diagonal entries are u_i's and p_i's, respectively. Then, the Leontief Economy Equilibrium $p \in R^n$, together with $u \in R^n$, must satisfy

$$\begin{array}{rcl} UHp & = & p, \\ P(e - H^{\mathrm{T}}u) & = & 0, \\ H^{\mathrm{T}}u & \le & e, \\ u,\ p & \ge & 0, \\ p & \ne & 0. \end{array} \tag{3}$$

One can prove:

Theorem 1 (Ye [8]) *Equation (3) always has a solution (u, p) under Assumption 1 (i. e., H has no all-zero row). However, a solution to Eq. (3) may not be a Leontief equilibrium, although every Leontief equilibrium satisfies Eq. (3).*

Theorem 2 (Ye [8]) *Let $B \subset \{1, 2, \ldots, n\}$, $N = \{1, 2, \ldots, n\} \setminus B$, H_{BB} be irreducible, and u_B satisfy the linear system*

$$H_{BB}^{\mathrm{T}} u_B = e, \quad H_{BN}^{\mathrm{T}} u_B \le e, \quad \text{and} \quad u_B > 0\,.$$

Then the (right) Perron–Frobenius eigen-vector p_B of $U_B H_{BB}$ together with $p_N = 0$ will be a solution to Eq. (3). And the converse is also true. Moreover, there is always a rational solution for every such B, that is, the entries of price vector are rational numbers, if the entries of H are rational. Furthermore, the size (bit-length) of the solution is bounded by the size (bit-length) of H.

The theorem implies that the traders in block B can trade among themselves and keep others goods "free". In particular, if one trader likes his or her own good more than any other good, that is, $h_{ii} \ge h_{ij}$ for all j, then $u_i = 1/h_{ii}$, $p_i = 1$, and $u_j = p_j = 0$ for all $j \ne i$, that is, $B = \{i\}$, makes a Leontief economy equilibrium. The theorem thus establishes, for the first time, a combinatorial algorithm to compute a Leontief economy equilibrium by finding a right block $B \ne \emptyset$, which is actually a non-trivial solution ($u \ne 0$) to an LCP problem

$$H^{\mathrm{T}}u + v = e,\ u^{\mathrm{T}}v = 0,\ 0 \ne u, v \ge 0. \tag{4}$$

If $H > 0$, then any complementary solution $u \ne 0$, together with its support $B = \{j\colon\ u_j > 0\}$, of Eq. (4) induce a Leontief economy equilibrium that is the (right) Perron–Frobenius eigen-vector of $U_B H_{BB}$, and it can be computed in polynomial time by solving a linear equation. Even if $H \not> 0$, any complementary solution $u \ne 0$ and $B = \{j\colon\ u_j > 0\}$, as long as H_{BB} is irreducible, induces an equilibrium for Eq. (3). The equivalence between the pairing Leontief economy model and the LCP also implies

Corollary 1 *LCP (4) always has a non-trivial solution, where H_{BB} is irreducible with $B = \{j\colon\ u_j > 0\}$, under Assumption 1 (i. e., H has no all-zero row).*

If Assumption 1 does not hold, the corollary may not be true; see example below:

$$H^{\mathrm{T}} = \begin{pmatrix} 0 & 2 \\ 0 & 1 \end{pmatrix}.$$

Applications

Given an arbitrary bimatrix game, specified by a pair of $n \times m$ matrices A and B, with positive entries, one can construct a Leontief exchange economy with $n + m$ traders and $n + m$ goods as follows. In words, trader i comes to the market with one unit of good i, for $i = 1, \ldots, n + m$.

Traders indexed by any $j \in \{1, \ldots, n\}$ receive some utility only from goods $j \in \{n+1, \ldots, n+m\}$, and this utility is specified by parameters corresponding to the entries of the matrix B. More precisely the proportions in which the j-th trader wants the goods are specified by the entries on the jth row of B. Vice versa, traders indexed by any $j \in \{n+1, \ldots, n+m\}$ receive some utility only from goods $j \in \{1, \ldots, n\}$. In this case, the proportions in which the j-th trader wants the goods are specified by the entries on the j-th column of A.

In the economy above, one can partition the traders in two groups, which bring to the market disjoint sets of goods, and are only interested in the goods brought by the group they do not belong to.

Theorem 3 (Codenotti et al. [4]) *Let (A,B) denote an arbitrary bimatrix game, where assume, w.l.o.g., that the entries of the matrices A and B are all positive. Let*

$$H^{\mathrm{T}} = \begin{pmatrix} 0 & A \\ B^{\mathrm{T}} & 0 \end{pmatrix}$$

describe the Leontief utility coefficient matrix of the traders in a Leontief economy. There is a one-to-one correspondence between the Nash equilibria of the game (A,B) and the market equilibria H of the Leontief economy. Furthermore, the correspondence has the property that a strategy is played with positive probability at a Nash equilibrium if and only if the good held by the corresponding trader has a positive price at the corresponding market equilibrium.

Gilboa and Zemel [6] proved a number of hardness results related to the computation of Nash equilibria (NE) for finite games in normal form. Since the NE for games with more than two players can be irrational, these results have been formulated in terms of NP-hardness for multiplayer games, while they can be expressed in terms of NP-completeness for two-player games. Using a reduction to the NE game, Codenotti et al. proved:

Theorem 4 (Codenotti et al. [4]) *It is NP-hard to decide whether a Leontief economy H has an equilibrium.*

Cross References

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

Recommended Reading

The reader may want to read Brainard and Scarf [2] on how to compute equilibrium prices in 1891; Chen and Deng [3] on the most recent hardness result of computing the bimatrix game; Cottle et al. [5] for literature on linear complementarity problems; and all references listed in [4] and [8] for the recent literature on computational equilibrium.

1. Arrow, K.J., Debreu, G.: Existence of an equilibrium for competitive economy. Econometrica **22**, 265–290 (1954)
2. Brainard, W.C., Scarf, H.E.: How to compute equilibrium prices in 1891. Cowles Foundation Discussion Paper 1270, August 2000
3. Chen, X., Deng, X.: Settling the complexity of 2-player Nash-Equilibrium, ECCC TR05-140 (2005)
4. Codenotti, B., Saberi, A., Varadarajan, K., Ye, Y.: Leontief economies encode nonzero sum two-player games. SODA (2006)
5. Cottle, R., Pang, J.S., Stone, R.E.: The linear complementarity problem. Academic Press, Boston (1992)
6. Gilboa, I., Zemel, E.: Nash and correlated equilibria: some complexity considerations. Games Econ. Behav. **1**, 80–93 (1989)
7. Walras, L.: Elements of pure economics, or the theory of social wealth (1899, 4th ed; 1926, rev ed, 1954, Engl. Transl.) (1874)
8. Ye, Y.: Exchange market equilibria with leontief's utility: freedom of pricing leads to rationality. WINE (2005)

Linearity Testing/ Testing Hadamard Codes

1990; Blum, Luby, Rubinfeld

Ronitt Rubinfeld
Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, USA

Keywords and Synonyms

Linearity testing; Testing Hadamard codes; Homomorphism testing

Problem Definition

This problem is concerned with distinguishing functions that are homomorphisms, i. e. satisfying $\forall x, y, f(x) + f(y) = f(x + y)$, from those functions that must be changed on at least ϵ fraction of the domain in order to be turned into a homomorphism, given query access to the function. This problem was initially motivated by applications to testing programs which compute linear functions [8]. Since Hadamard codes are such that the codewords are exactly the evaluations of linear functions over boolean variables, a solution to this problem gives a way of distinguishing codewords of the Hadamard code from those strings that are far in relative Hamming distance from codewords. These algorithms were in turn used in the constructions of Probabilistically Checkable Proof Sys-

tems (cf. [3]). Further work has extended these techniques to testing other properties of low degree polynomials and solutions to other addition theorems [3,9,24,25].

Notations

For two finite groups G, H (not necessarily Abelian), an arbitrary map $f : G \to H$ is a *homomorphism* if

$$\forall x, y,\ f(x) \times f(y) = f(x \times y)\ .$$

f is *ϵ-close to a homomorphism* if there is some homomorphism g such that g and f differ on at most $\epsilon|G|$ elements of G, and f is *ϵ-far* otherwise.

Given a parameter $0 \leq \epsilon \leq 1$, and query access to a function $f : G \to H$, a homomorphism tester (also refereed to as a *linearity tester* in the literature) is an algorithm $\mathcal{T}$ which outputs "Pass" if f is a homomorphism, and "Fail" if f is ϵ-far from a homomorphism. The homomorphism tester should err with probability at most 1/3 for any f.[1]

For two finite groups G, H (not necessarily Abelian), an arbitrary map $f : G \to H$, and a parameter $0 < \epsilon < 1$, define δ_f (the subscript is dropped and this is referred to as δ, when f is obvious from the context) then the *probability of group law failure*, by

$$1 - \delta = \Pr_{x,y}\left[f(x) \times f(y) = f(x \times y)\right]\ .$$

Define τ such that τ is the minimum ϵ for which f is ϵ-close to a homomorphism.

Problem 1 *For f, δ as above, is it possible to upper bound τ in terms of a function that depends on the probability of group law failure δ, but not on the size of the domain $|G|$?*

Key Results

Blum, Luby and Rubinfeld [8], considered this question and showed that over cyclic groups, there is a constant δ_0, such that if $\delta \leq \delta_0$, then the one can upper bound ϵ in terms of a function of δ that is independent of $|G|$. This yields a homomorphism tester with query complexity that depends (polynomially) on $1/\epsilon$, but is independent of $|G|$. The final version of [8] contains an improved argument due to Coppersmith [10], which applies to all Abelian groups, shows that $\delta_0 < 2/9$ suffices, and that ϵ is upper bounded by the smaller root of $x(1-x) = \delta$ (yielding a homomorphism tester with query complexity linear in $1/\epsilon$). Furthermore, the bound on δ_0 was shown to be tight for general groups [10].

In [6], a relationship between the probability of group law failure and the closeness to being a homomorphism was established that applies to general (non-Abelian) groups. For a given δ, let $\tau = (3 - \sqrt{9 - 24\delta})/12 \leq \delta/2$ be the smaller root of $3x - 6x^2 = \delta$. In [6] it is shown that for $\delta_0 < 2/9$, then f is τ-close to a homomorphism. The condition on δ, and the bound on τ as a function of δ, are shown to be tight. The latter improves on the relationship given in [8,10].

There has been interest in improving various parameters of homomorphism testing results, due to their use in the construction of Probabilistically Checkable Proof Systems (cf. [3]). In particular, both the constant δ_0 and the number of random bits required by the homomorphism test affect the efficiency of the proof system and in turn the hardness of approximation results that one can achieve using the proof system.

The homomorphism testing results can be improved in some cases: It has been previously mentioned that $\delta_0 < 2/9$ is optimal over general Abelian groups [10]. However, using Fourier techniques, Bellare et al. [5] have shown that for groups of the form $(\mathbf{Z}/2)^n$, $\delta_0 \leq 45/128$ suffices. For such δ_0, $\epsilon < \delta$. Kiwi later provided a similar result based on the discrete Fourier transform and weight distributions to improve the bound on the dependence of ϵ on δ [17].

Several works have shown methods of reducing the number of random bits required by the homomorphism tests. That is, in the natural implementation of the homomorphism test, $2\log|G|$ random bits per trial are used to pick x,y. The results of [7,14,26,28,29] have shown that fewer random bits are sufficient for implementing the homomorphism tests. In particular, Trevisan [29] and Samorodnitsky and Trevisan [26] have considered the "amortized query complexity" of testing homomorphisms, which is a measure that quantifies the trade-off between the query complexity of the testing algorithm and the probability of accepting the function. Homomorphism tests with low amortized query complexity are useful in constructing PCP systems with low amortized query complexity. A simpler analysis which improves the dependence of the acceptance probability in terms of the distance of the tested function to the closest linear function is given in [14]. The work of [28] gives a homomorphism test for general (non-Abelian) groups that uses only $(1 + o(1))\log_2|G|$ random bits. Given a Cayley graph that is an expander with normalized second eigenvalue γ, and

[1] The choice of 1/3 is arbitrary. Using standard techniques, any homomorphism tester satisfying 1/3 error probability can be turned into a homomorphism tester with $0 < \beta < 1/3$ error probability by repeating the original tester $O(\log \frac{1}{\beta})$ times and taking the majority answer.

for the analogous definitions of δ, τ, they show that for $\delta < (1-\gamma)/12$, τ is upper bounded by $4\delta/(1-\gamma)$. Very recently, Samordnitsky and Trevisan [27] have considered a relaxed version of a homomorphism test which accepts linear functions and rejects functions with low influences.

The case when G is a subset of an infinite group, f is a real-valued function and the oracle query to f returns a finite precision approximation to $f(x)$ has been considered in [2,11,12,20,21], and testers with query complexity that are independent of the domain size have been given (see [19] for a survey).

A Related Problem on Convolutions of Distributions

In the following, a seemingly unrelated question about distributions that are close to their self-convolutions is mentioned: Let $A = \{a_g | g \in G\}$ be a distribution on group G. The convolution of distributions A, B is

$$C = A * B,\ c_x = \sum_{y,z \in G;\ yz=x} a_y b_z\ .$$

Let A' be the *self-convolution* of A, $A * A$, i. e. $a'_x = \sum_{y,z \in G; yz=x} a_y a_z$. It is known that $A = A'$ exactly when A is the uniform distribution over a subgroup of G. Suppose it is known that A is close to A', can one say anything about A in this case? Suppose $dist(A, A') = \frac{1}{2}\sum_{x \in G} |a_x - a'_x| \leq \epsilon$ for small enough ϵ. Then [6] show that A must be close to the uniform distribution over a subgroup of G. More precisely, in [6] it is shown that for a distribution A over a group G, if $dist(A, A') = \frac{1}{2}\sum_{x \in G} |a_x - a'_x| \leq \epsilon \leq 0.0273$, then there is a subgroup H of G such that $dist(A, U_H) \leq 5\epsilon$, where U_H is the uniform distribution over H [6]. On the other hand, in [6] there is an example of a distribution A such that $dist(A, A * A) \approx .1504$, but A is not close to uniform on any subgroup of the domain.

A weaker version of this result, was used to prove a preliminary version of the homomorphism testing result in [8]. To give a hint of why one might consider the question on convolutions of distributions when investigating homomorphism testing, consider the distribution A_f achieved by picking x uniformly from G and outputting $f(x)$. It is easy to see that the error probability δ in the homomorphism test is at least $dist(A_f, A_f * A_f)$. The other, more useful, direction is less obvious. In [6] it is shown that this question on distributions is "equivalent" in difficulty to homomorphism testing:

Theorem 1 *Let G, H be finite groups. Assume that there is a parameter β_0 and function ϕ such that the following holds:*

> *For all distributions A over group G, if $dist(A * A, A) \leq \beta \leq \beta_0$ then A is $\phi(\beta)$-close to uniform over a subgroup of G.*

Then, for any $f : G \to H$ and $\delta < \beta_0$ such that $1 - \delta = Pr[f(x) * f(y) = f(x * y)]$, and $\phi(\delta) \leq 1/2$, it is the case that f is $\phi(\delta)$-close to a homomorphism.

Applications

Self-Testing/Correcting Programs

When a program has not been verified and therefore is not known to be correct on all inputs (or possibly even known to be incorrect on some inputs), [8] have suggested the following approach: take programs that are known to be correct on most inputs and apply a simple transformation to produce a program that is correct on every input. This transformation is referred to as producing a *self-corrector*. Moreover, for many functions, one can actually test that the program for f is correct on most inputs, without the aid of another program for f that has already been verified. Such testers for programs are referred to as *self-testers*.

The homomorphism testing problem was initially motivated by applications to constructing self-testers for programs which purport to compute various linear functions [8]. Such functions include integer, polynomial, matrix and modular multiplication and division. Once it is verified that a program agrees on most inputs with a specific linear function, the task of determining whether it agrees with the *correct* linear function on most inputs becomes much easier.

Furthermore, for programs purporting to compute linear functions, it is very simple to construct self-correctors: Assume one is given a program which on input x outputs $f(x)$, such that f agrees on most inputs with linear function g. Consider the algorithm that picks $c \log 1/\beta$ values y, computes $f(x+y) - f(y)$ and outputs the value that is seen most often. If f is $\frac{1}{8}$-close to g, then since both y and $x + y$ are uniformly distributed, it is the case that for at least 3/4 of the choices of y, $g(x+y) = f(x+y)$ *and* $g(y) = f(y)$, in which case $f(x+y) - f(y) = g(x)$. Thus it is easy to show that there is a constant c such that if f is $\frac{1}{8}$-close to a homomorphism g, then for all x, the above algorithm will output $g(x)$ with probability at least $1 - \beta$.

Probabilistically Checkable Proofs

An equivalent formulation of the homomorphism testing problem is in terms of the query complexity of testing a codeword of a Hadamard code. The results mentioned

about have been used to construct Probabilistically Checkable Proof systems which can be verified with very few queries (cf. [3,13]).

Open Problems

It is natural to wonder what other classes of functions have testers whose efficiency is sublinear in the domain size? There are some partial answers to this question: The field of functional equations is concerned with the prototypical problem of characterizing the set of functions that satisfy a given set of properties (or functional equations). For example, the class of functions of the form $f(x) = \tan Ax$ are characterized by the functional equation

$$\forall x, y, f(x+y) = \frac{f(x)+f(y)}{1-f(x)f(y)} .$$

D'Alembert's equation

$$\forall x, y, f(x+y) + f(x-y) = 2f(x)f(y)$$

characterizes the functions $0, \cos Ax, \cosh Ax$. Multivariate polynomials of total degree d over $\mathcal{Z}_p$ for $p > md$ can be characterized by the equation

$$\forall \hat{x}, \hat{h} \in Z_p^m, \sum_{i=0}^{d+1} \alpha_i f(\hat{x} + i\hat{h}) = 0 ,$$

where $\alpha_i = (-1)^{i+1}\binom{d+1}{i}$. All of the above characterizations are known to yield testers for the corresponding function families whose query complexity is independent of the domain size (though for the case of polynomials, there is a polynomial dependence on the total degree d) [9,24,25]. A long series of works have given increasingly efficient to test characterizations of functions that are low total degree polynomials (cf. [1,3,4,15,18,22,23]).

A second line of questions that remain to be understood regard which codes are such that strings can be quickly tested to determine whether they are close to a codeword? Some initial work on this questions is given in [1,15,16,18].

Cross References

► Learning Heavy Fourier Coefficients of Boolean Functions

Recommended Reading

1. Alon, N., Kaufman, T., Krivilevich, M., Litsyn, S., Ron, D.: Testing low-degree polynomials over gf(2). In: Proceedings of RANDOM '03. Lecture Notes in Computer Science, vol. 2764, pp. 188–199. Springer, Berlin Heidelberg (2003)
2. Ar, S., Blum, M., Codenotti, B., Gemmell, P.: Checking approximate computations over the reals. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing, pp. 786–795. ACM, New York (2003)
3. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. J. ACM **45**(3), 501–555 (1998)
4. Arora, S., Sudan, M.: Improved low degree testing and its applications. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, pp. 485–495. ACM, New York (1997)
5. Bellare, M., Coppersmith, D., Håstad, J., Kiwi, M., Sudan, M.: Linearity testing over characteristic two. IEEE Trans. Inf. Theory **42**(6), 1781–1795 (1996)
6. Ben-Or, M., Coppersmith, D., Luby, M., Rubinfeld, R.: Non-abelian homomorphism testing, and distributions close to their self-convolutions. In: Proceedings of APPROX-RANDOM. Lecture Notes in Computer Science, vol. 3122, pp. 273–285. Springer, Berlin Heidelberg (2004)
7. Ben-Sasson, E., Sudan, M., Vadhan, S., Wigderson, A.: Randomness-efficient low degree tests and short pcps via epsilon-biased sets. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on the Theory of Computing, pp. 612–621. ACM, New York (2003)
8. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. J. CSS **47**, 549–595 (1993)
9. Cleve, R., Luby, M.: A note on self-testing/correcting methods for trigonometric functions. In: International Computer Science Institute Technical Report TR-90-032, July 1990
10. Coppersmith, D.: Manuscript, private communications (1989)
11. Ergun, F., Kumar, R., Rubinfeld, R.: Checking approximate computations of polynomials and functional equations. SIAM J. Comput. **31**(2), 550–576 (2001)
12. Gemmell, P., Lipton, R., Rubinfeld, R., Sudan, M., Wigderson, A.: Self-testing/correcting for polynomials and for approximate functions. In: Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing, pp. 32–42. ACM, New York (1991)
13. Hastad, J.: Some optimal inapproximability results. J. ACM **48**(4), 798–859 (2001)
14. Hastad, J., Wigderson, A.: Simple analysis of graph tests for linearity and pcp. Random Struct. Algorithms **22**(2), 139–160 (2003)
15. Jutla, C., Patthak, A., Rudra, A., Zuckerman, D.: Testing low-degree polynomials over prime fields. In: Proceedings of the Forty-Fifth Annual Symposium on Foundations of Computer Science, pp. 423–432. IEEE, New York (2004)
16. Kaufman, T., Litsyn, S.: Almost orthogonal linear codes are locally testable. In: Proceedings of the Forty-Sixth Annual Symposium on Foundations of Computer Science, pp. 317–326. IEEE, New York (2005)
17. Kaufman, T., Litsyn, S., Xie, N.: Breaking the ϵ-soundness bound of the linearity test over gf(2). Electronic Colloquium on Computational Complexity, Report TR07–098, October 2007
18. Kaufman, T., Ron, D.: Testing polynomials over general fields. In: Proceedings of the Forty-Fifth Annual Symposium on Foundations of Computer Science, pp. 413–422. IEEE, New York (2004)
19. Kiwi, M., Magniez, F., Santha, M.: Exact and approximate testing/correcting of algebraic functions: A survey. Theoretical Aspects Compututer Science, LNCS **2292**, 30–83 (2001)

20. Kiwi, M., Magniez, F., Santha, M.: Approximate testing with error relative to input size. J. CSS **66**(2), 371–392 (2003)
21. Magniez, F.: Multi-linearity self-testing with relative error. Theory Comput. Syst. **38**(5), 573–591 (2005)
22. Polischuk, A., Spielman, D.: Nearly linear-size holographic proofs. In: Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing, pp. 194–203. ACM, New York (1994)
23. Raz, R., Safra, S.: A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, pp. 475–484. ACM, New York (1997)
24. Rubinfeld, R.: On the robustness of functional equations. SIAM J. Comput. **28**(6), 1972–1997 (1999)
25. Rubinfeld, R., Sudan, M.: Robust characterization of polynomials with applications to program testing. SIAM J. Comput. **25**(2), 252–271 (1996)
26. Samorodnitsky, A., Trevisan, L.: A PCP characterization of NP with optimal amortized query complexity. In: Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing, pp. 191–199. ACM, New York (2000)
27. Samorodnitsky, A., Trevisan, L.: Gowers uniformity, influence of variables, and pcps. In: Thirty-Eighth ACM Symposium on Theory of Computing, pp. 11–20. ACM, New York (2006)
28. Shpilka, A., Wigderson, A.: Derandomizing homomorphism testing in general groups. In: Proceedings of the Thirty-Sixth Annual ACM Symposium on the Theory of Computing, pp. 427–435. ACM, NY, USA (2004)
29. Trevisan, L.: Recycling queries in pcps and in linearity tests. In: Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, pp. 299–308. ACM, New York (1998)

Linearizability

1990; Herlihy, Wing

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

Keywords and Synonyms

Atomicity

Problem Definition

An *object* in languages such as Java and C++ is a container for data. Each object provides a set of *methods* that are the only way to to manipulate that object's internal state. Each object has a *class* which defines the methods it provides and what they do.

In the absence of concurrency, methods can be described by a pair consisting of a *precondition* (describing the object's state before invoking the method) and a *post-condition*, describing, once the method returns, the object's state and the method's return value. If, however, an object is shared by concurrent threads in a multiprocessor system, then method calls may overlap in time, and it no longer makes sense to characterize methods in terms of pre- and post-conditions.

Linearizability is a correctness condition for concurrent objects that characterizes an object's concurrent behavior in terms of an "equivalent" sequential behavior. Informally, the object behaves "as if" each method call takes effect instantaneously at some point between its invocation and its response. This notion of correctness has some useful formal properties. First, it is *non-blocking*, which means that linearizability as such never requires one thread to wait for another to complete an ongoing method call. Second, it is *local*, which means that an object composed of linearizable objects is itself linearizable. Other proposed correctness conditions in the literature lack at least one of these properties.

Notation

An execution of a concurrent system is modeled by a *history*, a finite sequence of method *invocation* and *response events*. A *subhistory* of a history H is a subsequence of the events of H. A method invocation is written as $\langle x.m(a^*)A\rangle$, where x is an object, m a method name, a^* a sequence of arguments, and A a thread. A method response is written as $\langle x\colon t(r^*)A\rangle$ where t is a termination condition and r^* is a sequence of result values.

A response *matches* an invocation if their objects and thread names agree. A *method call* is a pair consisting of an invocation and the next matching response. An invocation is *pending* in a history if no matching response follows the invocation. If H is a history, *complete*(H) is the subsequence of H consisting of all matching invocations and responses. A history H is *sequential* if the first event of H is an invocation, and each invocation, except possibly the last, is immediately followed by a matching response.

Let H be a a history. The *thread subhistory* $H|P$ is the subsequence of events in H with thread name P. The *object subhistory* $H|x$ is similarly defined for an object x. Two histories H and H' are *equivalent* if for every thread A, $H|A = H'|A$. A history H is *well-formed* if each thread subhistory $H|A$ of H is sequential. Notice that thread subhistories of a well-formed history are always sequential, but object subhistories need not be.

A *sequential specification* for an object is a prefix-closed set of sequential object histories that defines that object's *legal* histories. A sequential history H is *legal* if each object subhistory is legal. A method is *total* if it is defined for every object state, otherwise it is *partial*. (For example, a *deq()* method that blocks on an empty queue is partial, while one that throws an exception is total.)

A history H defines an (irreflexive) partial order $\rightarrow_H$ on its method calls: $m_0 \rightarrow_H m_1$ if the result event of m_0 occurs before the invocation event of m_1. If H is a sequential history, then $\rightarrow_H$ is a total order.

Let H be a history and x an object such that $H|x$ contains method calls m_0 and m_1. A call $m_0 \rightarrow_x m_1$ if m_0 precedes m_1 in $H|x$. Note that $\rightarrow_x$ is a total order.

Informally, linearizability requires that each method call appear to "take effect" instantaneously at some moment between its invocation and response. An important implication of this definition is that method calls that do not overlap cannot be reordered: linearizability preserves the "real-time" order of method calls. Formally,

Definition 1 A history H is *linearizable* if it can be extended (by appending zero or more response events) to a history H' such that:

- L1 $complete(H')$ is equivalent to a legal sequential history S, and
- L2 If method call m_0 precedes method call m_1 in H, then the same is true in S.

S is called a *linearization* of H. (H may have multiple linearizations.) Informally, extending H to H' captures the idea that some pending invocations may have taken effect even though their responses have not yet been returned to the caller.

Key Results

The Locality Property

A property is *local* if all objects collectively satisfy that property provided that each individual object satisfies it.

Linearizability is local:

Theorem 1 *H is linearizable if and only if $H|x$ is linearizable for ever object x.*

Proof The "only if" part is obvious.

For each object x, pick a linearization of $H|x$. Let R_x be the set of responses appended to $H|x$ to construct that linearization, and let $\rightarrow_x$ be the corresponding linearization order. Let H' be the history constructed by appending to H each response in R_x.

The $\rightarrow_H$ and $\rightarrow_x$ orders can be "rolled up" into a single partial order. Define the relation $\rightarrow$ on method calls of $complete(H')$: For method calls m and $\bar{m}$, $m \rightarrow \bar{m}$ if there exist method calls $m_0, \ldots, m_n$, such that $m = m_0, \bar{m} = m_n$, and for each i between 0 and $n-1$, either $m_i \rightarrow_x m_{i+1}$ for some object x, or $m_i \rightarrow_H m_{i+1}$.

It turns out that $\rightarrow$ is a partial order. Clearly, $\rightarrow$ is transitive. It remains to be shown that $\rightarrow$ is anti-reflexive: for all x, it is false that $x \rightarrow x$.

The proof proceeds by contradiction. If not, then there exist method calls $m_0, \ldots, m_n$, such that $m_0 \rightarrow m_1 \rightarrow \cdots \rightarrow m_n, m_n \rightarrow m_0$, and each pair is directly related by some $\rightarrow_x$ or by $\rightarrow_H$.

Choose a cycle whose length is minimal. Suppose all method calls are associated with the same object x. Since $\rightarrow_x$ is a total order, there must exist two method calls m_{i-1} and m_i such that $m_{i-1} \rightarrow_H m_i$ and $m_i \rightarrow_x m_{i-1}$, contradicting the linearizability of x.

The cycle must therefore include method calls of at least two objects. By reindexing if necessary, let m_1 and m_2 be method calls of distinct objects. Let x be the object associated with m_1. None of $m_2, \ldots, m_n$ can be a method call of x. The claim holds for m_2 by construction. Let m_i be the first method call in $m_3, \ldots, m_n$ associated with x. Since m_{i-1} and m_i are unrelated by $\rightarrow_x$, they must be related by $\rightarrow_H$, so the response of m_{i-1} precedes the invocation of m_i. The invocation of m_2 precedes the response of m_{i-1}, since otherwise $m_{i-1} \rightarrow_H m_2$, yielding the shorter cycle $m_2, \ldots, m_{i-1}$. Finally, the response of m_1 precedes the invocation of m_2, since $m_1 \rightarrow_H m_2$ by construction. It follows that the response to m_1 precedes the invocation of m_i, hence $m_1 \rightarrow_H m_i$, yielding the shorter cycle $m_1, m_i, \ldots, m_n$.

Since m_n is not a method call of x, but $m_n \rightarrow m_1$, it follows that $m_n \rightarrow_H m_1$. But $m_1 \rightarrow_H m_2$ by construction, and because $\rightarrow_H$ is transitive, $m_n \rightarrow_H m_2$, yielding the shorter cycle $m_2, \ldots, m_n$, the final contradiction. □

Locality is important because it allows concurrent systems to be designed and constructed in a modular fashion; linearizable objects can be implemented, verified, and executed independently. A concurrent system based on a non-local correctness property must either rely on a centralized scheduler for all objects, or else satisfy additional constraints placed on objects to ensure that they follow compatible scheduling protocols. Locality should not be taken for granted; as discussed below, the literature includes proposals for alternative correctness properties that are not local.

The Non-Blocking Property

Linearizability is a *non-blocking* property: a pending invocation of a total method is never required to wait for another pending invocation to complete.

Theorem 2 *Let inv(m) be an invocation of a total method. If $\langle x\ invP\rangle$ is a pending invocation in a linearizable history H, then there exists a response $\langle x\ resP\rangle$ such that $H \cdot \langle x\ resP\rangle$ is linearizable.*

Proof Let S be any linearization of H. If S includes a response $\langle x\ resP\rangle$ to $\langle x\ invP\rangle$, the proof is complete, since S is also a linearization of $H \cdot \langle x\ resP\rangle$. Otherwise, $\langle x\ invP\rangle$ does not appear in S either, since linearizations, by definition, include no pending invocations. Because the method is total, there exists a response $\langle x\ resP\rangle$ such that

$$S' = S \cdot \langle x\ invP\rangle \cdot \langle x\ res\ P\rangle$$

is legal. S', however, is a linearization of $H \cdot \langle x\ resP\rangle$, and hence is also a linearization of H. □

This theorem implies that linearizability by itself never forces a thread with a pending invocation of a total method to block. Of course, blocking (or even deadlock) may occur as artifacts of particular implementations of linearizability, but it is not inherent to the correctness property itself. This theorem suggests that linearizability is an appropriate correctness condition for systems where concurrency and real-time response are important. Alternative correctness conditions, such as serializability [1] do not share this non-blocking property.

The non-blocking property does not rule out blocking in situations where it is explicitly intended. For example, it may be sensible for a thread attempting to dequeue from an empty queue to block, waiting until another thread enqueues an item. The queue specification captures this intention by making the `deq()` method's specification partial, leaving it's effect undefined when applied to an empty queue. The most natural concurrent interpretation of a partial sequential specification is simply to wait until the object reaches a state in which the method is defined.

Other Correctness Properties

Sequential Consistency [4] is a weaker correctness condition that requires Property *L1* but not *L2*: method calls must appear to happen in some one-at-a-time, sequential order, but calls that do not overlap can be reordered. Every linearizable history is sequentially consistent, but not vice versa. Sequential consistency permits more concurrency, but it is not a local property: a system composed of multiple sequentially-consistent objects is not itself necessarily sequentially consistent.

Much work on databases and distributed systems uses *serializability* as the basic correctness condition for concurrent computations. In this model, a *transaction* is a "thread of control" that applies a finite sequence of methods to a set of objects shared with other transactions. A history is *serializable* if it is equivalent to one in which transactions appear to execute sequentially, that is, without interleaving. A history is *strictly serializable* if the transactions' order in the sequential history is compatible with their precedence order: if every method call of one transaction precedes every method call of another, the former is serialized first. (Linearizability can be viewed as a special case of strict serializability where transactions are restricted to consist of a single method applied to a single object.)

Neither serializability nor strict serializability is a local property. If different objects serialize transactions in different orders, then there may be no serialization order common to all objects. Serializability and strict serializability are *blocking* properties: Under certain circumstances, a transaction may be unable to complete a pending method without violating serializability. A *deadlock* results if multiple transactions block one another. Such transactions must be rolled back and restarted, implying that additional mechanisms must be provided for that purpose.

Applications

Linearizability is widely used as the basic correctness condition for many concurrent data structure algorithms [5], particularly for lock-free and wait-free data structures [2]. Sequential consistency is widely used for describing low-level systems such as hardware memory interfaces. Serializability and strict serializability are widely used for database systems in which it must be easy for application programmers to preserve complex application-specific invariants spanning multiple objects.

Open Problems

Modern multiprocessors often support very weak models of memory consistency. There are many open problems concerning how to model such behavior, and how to ensure linearizable object implementations on top of such architectures.

Cross References

► Concurrent Programming, Mutual Exclusion
► Registers

Recommended Reading

The notion of Linearizability is due to Herlihy and Wing [3], while Sequential Consistency is due to Lamport [4], and serializability to Eswaran et al. [1].

1. Eswaran, K.P., Gray, J.N., Lorie, R.A., Traiger, I.L.: The notions of consistency and predicate locks in a database system. Commun. ACM **19**(11), 624–633 (1976). doi: http://doi.acm.org/10.1145/360363.360369

2. Herlihy, M.: Wait-free synchronization. ACM Trans. Program. Lang. Syst. (TOPLAS) **13**(1), 124–149 (1991)
3. Herlihy, M.P., Wing, J.M.: Linearizability: a correctness condition for concurrent objects. ACM Trans. Program. Lang. Syst. (TOPLAS) **12**(3), 463–492 (1990)
4. Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Trans. Comput. **C-28**(9), 690 (1979)
5. Vafeiadis, V., Herlihy, M., Hoare, T., Shapiro, M.: Proving correctness of highly-concurrent linearisable objects. In: PPoPP '06: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming, pp. 129–136 (2006). doi: http://doi.acm.org/10.1145/1122971.1122992

List Decoding near Capacity: Folded RS Codes

2006; Guruswami, Rudra

Atri Rudra
Department of Computer Science and Engineering, University at Buffalo, State University of New York, Buffalo, NY, USA

Keywords and Synonyms

Decoding; Error correction

Problem Definition

One of the central trade-offs in the theory of error-correcting codes is the one between the amount of redundancy needed and the fraction of errors that can be corrected.[1] The redundancy is measured by the *rate* of the code, which is the ratio of the the number of information symbols in the message to that in the codeword – thus, for a code with encoding function $E : \Sigma^k \to \Sigma^n$, the rate equals k / n. The *block length* of the code equals n, and Σ is its *alphabet*.

The goal in *decoding* is to find, given a noisy received word, the actual codeword that it could have possibly resulted from. If the target is to correct a fraction ρ of errors (ρ will be called the error-correction radius), then this amounts to finding codewords within (normalized Hamming) distance ρ from the received word. We are guaranteed that there will be a unique such codeword provided the distance between *every* two distinct codewords is at least 2ρ, or in other words the relative distance of the code is at least 2ρ. However, since the relative distance δ of a code must satisfy $\delta \leq 1 - R$ where R is the rate of the code (by the Singleton bound), if one insists on an unique answer, the best trade-off between ρ and R is $\rho = \rho_U(R) = (1 - R)/2$. But this is an overly pessimistic estimate of the error-correction radius, since the way Hamming spheres pack in space, for *most* choices of the received word there will be at most one codeword within distance ρ from it even for ρ much greater than $\delta/2$. Therefore, *always* insisting on a unique answer will preclude decoding most such received words owing to a few pathological received words that have more than one codeword within distance roughly $\delta/2$ from them.

A notion called list decoding, that dates back to the late 1950's [1,9], provides a clean way to get around this predicament, and yet deal with worst-case error patterns. Under list decoding, the decoder is required to output a list of all codewords within distance ρ from the received word. Let us call a code C (ρ, L)-*list decodable* if the number of codewords within distance ρ of any received word is at most L. To obtain better trade-offs via list decoding, (ρ, L)-list decodable codes are needed where L is bounded by a polynomial function of the block length, since this an *a priori* requirement for polynomial time list decoding. How large can ρ be as a function of R for which such (ρ, L)-list decodable codes exist? A standard random coding argument shows that $\rho \geq 1 - R - o(1)$ can be achieved over large enough alphabets, cf. [2,10], and a simple counting argument shows that ρ must be at most $1 - R$. Therefore the *list decoding capacity*, i. e., the information-theoretic limit of list decodability, is given by the trade-off $\rho_{\text{cap}}(R) = 1 - R = 2\rho_U(R)$. Thus list decoding holds the promise of correcting *twice* as many errors as unique decoding, for *every* rate. The above-mentioned list decodable codes are non-constructive. In order to realize the potential of list decoding, one needs explicit constructions of such codes, and on top of that, polynomial time algorithms to perform list decoding.

Building on works of Sudan [8], Guruswami and Sudan [6] and Parvaresh and Vardy [7], Guruswami and Rudra [5] present codes that get arbitrarily close to the list decoding capacity $\rho_{\text{cap}}(R)$ for every rate. In particular, for every $1 > R > 0$ and every $\epsilon > 0$, they give *explicit* codes of rate R together with polynomial time list decoding algorithm that can correct up to a fraction $1 - R - \epsilon$ of errors. These are the first explicit codes (with efficient list decoding algorithms) that get arbitrarily close to the list decoding capacity for *any* rate.

Description of the Code

Consider a Reed–Solomon (RS) code $C = \mathsf{RS}_{\mathbb{F},\mathbb{F}^*}[n, k]$ consisting of evaluations of degree k polynomials over

[1] This entry deals with the adversarial or worst-case model of errors—no assumption is made on how the errors and error locations are distributed beyond an upper bound on the total number of errors that may be caused.

some finite field $\mathbb{F}$ at the set $\mathbb{F}^*$ of nonzero elements of $\mathbb{F}$. Let $q = |\mathbb{F}| = n + 1$. Let γ be a generator of the multiplicative group $\mathbb{F}^*$, and let the evaluation points be ordered as $1, \gamma, \gamma^2, \ldots, \gamma^{n-1}$. Using all nonzero field elements as evaluation points is one of the most commonly used instantiations of Reed–Solomon codes.

Let $m \geq 1$ be an integer parameter called the *folding parameter*. For ease of presentation, it will assumed that m divides $n = q - 1$.

Definition 1 (Folded Reed–Solomon Code) The m-folded version of the RS code C, denoted $\mathsf{FRS}_{\mathbb{F},\gamma,m,k}$, is a code of block length $N = n/m$ over $\mathbb{F}^m$. The encoding of a message $f(X)$, a polynomial over $\mathbb{F}$ of degree at most k, has as its j'th symbol, for $0 \leq j < n/m$, the m-tuple $(f(\gamma^{jm}), f(\gamma^{jm+1}), \cdots, f(\gamma^{jm+m-1}))$. In other words, the codewords of $C' = \mathsf{FRS}_{\mathbb{F},\gamma,m,k}$ are in one-one correspondence with those of the RS code C and are obtained by bundling together consecutive m-tuple of symbols in codewords of C.

Key Results

The following is the main result of Guruswami and Rudra.

Theorem 1 ([5]) *For every $\epsilon > 0$ and $0 < R < 1$, there is a family of folded Reed–Solomon codes that have rate at least R and which can be list decoded up to a fraction $1 - R - \epsilon$ of errors in time (and outputs a list of size at most) $(N/\epsilon^2)^{O(\epsilon^{-1}\log(1/R))}$ where N is the block length of the code. The alphabet size of the code as a function of the block length N is $(N/\epsilon^2)^{O(1/\epsilon^2)}$.*

The result of Guruswami and Rudra also works in a more general setting called *list recovering*, which is defined next.

Definition 2 (List Recovering) A code $C \subseteq \Sigma^n$ is said to be (ζ, l, L)-list recoverable if for every sequence of sets $S_1, \cdots, S_n$ where each $S_i \subseteq \Sigma$ has at most l elements, the number of codewords $c \in C$ for which $c_i \in S_i$ for at least ζn positions $i \in \{1, 2, \ldots, n\}$ is at most L.

A code $C \subseteq \Sigma^n$ is said to (ζ, l)-list recoverable in polynomial time if it is $(\zeta, l, L(n))$-list recoverable for some polynomially bounded function $L(\cdot)$, and moreover there is a polynomial time algorithm to find the at most $L(n)$ codewords that are solutions to any $(\zeta, l, L(n))$-list recovering instance.

Note that when $l = 1$, $(\zeta, 1, \cdot)$-list recovering is the same as list decoding up to a $(1 - \zeta)$ fraction of errors. Guruswami and Rudra have the following result for list recovering.

Theorem 2 ([5]) *For every integer $l \geq 1$, for all R, $0 < R < 1$ and $\epsilon > 0$, and for every prime p, there is an explicit family of folded Reed–Solomon codes over fields of characteristic p that have rate at least R and which can be $(R + \epsilon, l)$-list recovered in polynomial time. The alphabet size of a code of block length N in the family is $(N/\epsilon^2)^{O(\epsilon^{-2}\log l/(1-R))}$.*

Applications

To get within ϵ of capacity, the codes in Theorem 1 have alphabet size $N^{\Omega(1/\epsilon^2)}$ where N is the block length. By concatenating folded RS codes of rate close to 1 (that are list recoverable) with suitable inner codes followed by redistribution of symbols using an expander graph (similar to a construction for linear-time unique decodable codes in [3]), one can get within ϵ of capacity with codes over an alphabet of size $2^{O(\epsilon^{-4}\log(1/\epsilon))}$. A counting argument shows that codes that can be list decoded efficiently to within ϵ of the capacity need to have an alphabet size of $2^{\Omega(1/\epsilon)}$.

For binary codes, the list decoding capacity is known to be $\rho_{\text{bin}}(R) = H^{-1}(1 - R)$ where $H(\cdot)$ denotes the binary entropy function. No explicit constructions of binary codes that approach this capacity are known. However, using the Folded RS codes of Guruswami Rudra in a natural concatenation scheme, one can obtain polynomial time constructable binary codes of rate R that can be list decoded up to a fraction $\rho_{\text{Zyab}}(R)$ of errors, where $\rho_{\text{Zyab}}(R)$ is the "Zyablov bound".

See [5] for more details.

Open Problems

The work of Guruswami and Rudra could be improved with respect to some parameters. The size of the list needed to perform list decoding to a radius that is within ϵ of capacity grows as $N^{O(\epsilon^{-1}\log(1/R))}$ where N and R are the block length and the rate of the code respectively. It remains an open question to bring this list size down to a constant independent of n (the existential random coding arguments work with a list size of $O(1/\epsilon)$). The alphabet size needed to approach capacity was shown to be a constant independent of N. However, this involved a brute-force search for a rather large (inner) code, which translates to a construction time of about $N^{O(\epsilon^{-2}\log(1/\epsilon))}$ (instead of the ideal construction time where the exponent of N does not depend on ϵ). Obtaining a "direct" algebraic construction over a constant-sized alphabet, such as the generalization of the Parvaresh-Vardy framework to algebraic-geometric codes in [4], might help in addressing these two issues.

Finally, constructing binary codes that approach list decoding capacity remains open.

Cross References

- ▶ Decoding Reed–Solomon Codes
- ▶ Learning Heavy Fourier Coefficients of Boolean Functions
- ▶ LP Decoding

Recommended Reading

1. Elias, P.: List decoding for noisy channels. Technical Report 335, Research Laboratory of Electronics MIT (1957)
2. Elias, P.: Error-correcting codes for list decoding. IEEE Trans. Inf. Theory **37**, 5–12 (1991)
3. Guruswami, V., Indyk, P.: Linear-time encodable/decodable codes with near-optimal rate. IEEE Trans. Inf. Theory **51**(10), 3393–3400 (2005)
4. Guruswami, V., Patthak, A.: Correlated Algebraic-Geometric codes: Improved list decoding over bounded alphabets. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 227–236, Berkley, October 2006
5. Guruswami, V., Rudra, A.: Explicit capacity-achieving list-decodable codes. In: Proceedings of the 38th Annual ACM Symposium on Theory of Computing, pp. 1–10. Seattle, May 2006
6. Guruswami, V., Sudan, M.: Improved decoding of Reed–Solomon and algebraic-geometric codes. IEEE Trans. Inf. Theory **45**, 1757–1767 (1999)
7. Parvaresh, F., Vardy, A.: Correcting errors beyond the Guruswami—Sudan radius in polynomial time. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, pp. 285–294. Pittsburgh, 2005
8. Sudan, M.: Decoding of Reed—Solomon codes beyond the error-correction bound. J Complex. **13**(1), 180–193 (1997)
9. Wozencraft, J.M.: List Decoding. Quarterly Progress Report, Research Laboratory of Electronics. MIT **48**, 90–95 (1958)
10. Zyablov, V.V., Pinsker, M.S.: List cascade decoding. Probl. Inf. Trans. **17**(4), 29–34 (1981) (in Russian); pp. 236–240 (in English) (1982)

List Scheduling

1966; Graham

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

Keywords and Synonyms

Online scheduling on identical machines

Problem Definition

The paper of Graham [8] was published in the 1960s. Over the years it served as a common example of online algorithms (though the original algorithm was designed as a simple approximation heuristic). The following basic setting is considered.

A sequence of n jobs is to be assigned to m identical machines. Each job should be assigned to one of the machines. Each job has a size associated with it, which can be seen as its processing time or its load. The load of a machine is the sum of sizes of jobs assigned to it. The goal is to minimize the maximum load of any machine, also called the makespan. We refer to this problem as Job Scheduling.

If jobs are presented one by one, and each job needs to be assigned to a machine in turn, without any knowledge of future jobs, the problem is called online. Online algorithms are typically evaluated using the (absolute) *competitive ratio*, which is similar to the *approximation ratio* of approximation algorithms. For an algorithm $\mathcal{A}$, we denote its cost by $\mathcal{A}$ as well. The cost of an optimal offline algorithm that knows the complete sequence of jobs is denoted by OPT. The competitive ratio of an algorithm $\mathcal{A}$ is the infimum $\mathcal{R} \geq 1$ such that for any input, $\mathcal{A} \leq \mathcal{R} \cdot \mathrm{OPT}$.

Key Results

In paper [8], Graham defines an algorithm called List Scheduling (LS). The algorithm receives jobs one by one. Each job is assigned in turn to a machine which has a minimal current load. Ties are broken arbitrarily.

The main result is the following.

Theorem 1 LS *has a competitive ratio of* $2 - \frac{1}{m}$.

Proof. Consider a schedule created for a given sequence. Let ℓ denote a job that determines the makespan (that is, the last job assigned to a machine i that has a maximum load), let L denote its size, and let X denote the total size of all other jobs assigned to i. At the time when L was assigned to i, this was a machine of minimum load. Therefore, the load of each machine is at least X. The makespan of an optimal schedule (i. e., a schedule that minimizes the makespan) is the cost of an optimal offline algorithm and thus is denoted by OPT. Let P be the sum of all job sizes in the sequence.

The two following simple lower bounds on OPT can be obtained.

$$\mathrm{OPT} \geq L\,. \tag{1}$$

$$\mathrm{OPT} \geq \frac{P}{m} \geq \frac{m \cdot X + L}{m} = X + \frac{L}{m}\,. \tag{2}$$

Inequality (1) follows from the fact that {OPT} needs to run job ℓ and thus at least one machine has a load of at least L. The first inequality in (2) is due to the fact that at least one machine receives at least a fraction of $\frac{1}{m}$ of the total size of jobs. The second inequality in (2) follows from the comments above on the load of each machine.

This proves that the makespan of the algorithm, $X + L$ can be bounded as follows.

$$1X + L \leq \mathrm{OPT} + \frac{m-1}{m}L \leq \mathrm{OPT} + \frac{m-1}{m}\mathrm{OPT} = (2 - 1/m)\,\mathrm{OPT}\,. \quad (3)$$

The first inequality in (3) follows from (2) and the second one from (1).

To show that the analysis is tight, consider $m(m-1)$ jobs of size 1 followed by a single job of size m. After the smaller jobs arrive, LS obtains a balanced schedule in which every machine has a load of $m-1$. The additional job increases the makespan to $2m-1$. However, an optimal offline solution would be to assign the smaller jobs to $m-1$ machines, and the remaining job to the remaining machine, getting a load of m.

A natural question was whether this bound is best possible. In a later paper, Graham [9] showed that applying LS with a sorted sequence of jobs (by non-increasing order of sizes) actually gives a better upper bound of $\frac{4}{3} - \frac{1}{3m}$ on the approximation ratio. A polynomial time approximation scheme was given by Hochbaum and Shmoys in [10]. This is the best offline result one could hope for as the problem is known to be NP-hard in the strong sense.

As for the online problem, it was shown in [5] that no (deterministic) algorithm has a smaller competitive ratio than $2 - \frac{1}{m}$, for the cases $m = 2$ and $m = 3$. On the other hand, it was shown in a sequence of papers that an algorithm with a smaller competitive ratio can be found for any $m \geq 4$, and even algorithms with a competitive ratio that does not approach 2 for large m were designed.

The best such result is by Fleischer and Wahl [6], who designed a 1.9201-competitive algorithm. Lower bounds of 1.852 and 1.85358 on the competitive ratio of any online algorithm were shown in [1,7]. Rudin [13] claimed a better lower bound of 1.88.

Applications

As the study of approximation algorithms and specifically online algorithms continued, the analysis of many scheduling algorithms used similar methods to the proof above. Below, several variants of the problem where almost the same proof as above gives the exact same bound are mentioned.

Load Balancing of Temporary Tasks

In this problem the sizes of jobs are seen as loads. Time is a separate axis. The input is a sequence of events, where every event is an arrival or a departure of a job. The set of active jobs at time t is the set of jobs that have already arrived at this time and have not departed yet. The cost of an algorithm at a time t is its makespan at this time. The cost of an algorithm is its maximum cost over time. It turns out that the analysis above can be easily adapted for this model as well. It is interesting to note that in this case the bound $2 - \frac{1}{m}$ is actually best possible, as shown in [2].

Scheduling with Release Times and Precedence Constraints

In this problem, the sizes represent processing times of jobs. Various versions have been studied. Jobs may have designated release times, which are the times when these jobs become available for execution. In the online scenario, each job arrives and becomes known to the algorithm only at its release time. Some precedence constraints may also be specified, defined by a partial order on the set of jobs. Thus, a job can be run only after its predecessors complete their execution. In the online variant, a job becomes known to the algorithm only after its predecessors have been completed. In these cases, LS acts as follows. Once a machine becomes available, a waiting job that arrived earliest is assigned to it. (If there is no waiting job, the machine is idle until a new job arrives.)

The upper bound of $2 - \frac{1}{m}$ on the competitive ratio can be proved using a relation between the cost of an optimal schedule, and the amount of time when at least one machine is idle. (See [14] for details).

This bound is tight for several cases. For the case where there are release times, no precedence constraints, and processing times (sizes) are not known upon arrival, Shmoys, Wein, and Williamson [15] proved a lower bound of $2 - \frac{1}{m}$. For the case where there are only precedence constraints (no release times, and sizes of jobs are known upon arrival), a lower bound of the same value appeared in [4]. Note that the case with clairvoyant scheduling (i. e., sizes of jobs are known upon arrival), release times, and no precedence constraints is not settled. For $m = 2$ it was shown by Noga and Seiden [11] that the tight bound is $(5 - \sqrt{5})/2 \approx 1.38198$, and the upper bound is achieved using an algorithm that applies waiting with idle machines rather than scheduling a job as soon as possible, as done by LS.

Open Problems

The most challenging open problem is to find the best possible competitive ratio for this basic online problem of job scheduling. The gap between the upper bound and the lower bound is not large, yet it seems very difficult to find the exact bound. A possibly easier question would be to find the best possible competitive ratio for $m = 4$. A lower

bound of $\sqrt{3} \approx 1.732$ has been shown by [12] and the currently known upper bound is 1.733 by [3]. Thus, it may be the case that this bound would turn out to be $\sqrt{3}$.

Recommended Reading

1. Albers, S.: Better bounds for online scheduling. SIAM J. Comput. **29**(2), 459–473 (1999)
2. Azar, Y., Epstein, L.: On-line load balancing of temporary tasks on identical machines. SIAM J. Discret. Math. **18**(2), 347–352 (2004)
3. Chen, B., van Vliet, A., Woeginger, G.J.: New lower and upper bounds for on-line scheduling. Oper. Res. Lett. **16**, 221–230 (1994)
4. Epstein, L.: A note on on-line scheduling with precedence constraints on identical machines. Inf. Process. Lett. **76**, 149–153 (2000)
5. Faigle, U., Kern, W., Turán, G.: On the performane of online algorithms for partition problems. Acta Cybern. **9**, 107–119 (1989)
6. Fleischer, R., Wahl, M.: On-line scheduling revisited. J. Sched. **3**, 343–353 (2000)
7. Gormley, T., Reingold, N., Torng, E., Westbrook, J.: Generating adversaries for request-answer games. In: Proc. of the 11th Symposium on Discrete Algorithms. (SODA2000), pp. 564–565 (2000)
8. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell Syst. Techn. J. **45**, 1563–1581 (1966)
9. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. **17**, 263–269 (1969)
10. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: theoretical and practical results. J. ACM **34**(1), 144–162 (1987)
11. Noga, J., Seiden, S.S.: An optimal online algorithm for scheduling two machines with release times. Theor. Comput. Sci. **268**(1), 133–143 (2001)
12. Rudin III, J.F., Chandrasekaran, R.: Improved bounds for the online scheduling problem. SIAM J. Comput. **32**, 717–735 (2003)
13. Rudin III, J.F.: Improved bounds for the online scheduling problem. Ph. D. thesis, The University of Texas at Dallas (2001)
14. Sgall, J.: On-line scheduling. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms: The State of the Art, pp. 196–231. Springer (1998)
15. Shmoys, D.B., Wein, J., Williamson, D.P.: Scheduling parallel machines on-line. SIAM J. Comput. **24**, 1313–1331 (1995)

Load Balancing

1994; Azar, Broder, Karlin
1997; Azar, Kalyanasundaram, Plotkin, Pruhs, Waarts

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

Keywords and Synonyms

Online scheduling of temporary tasks

Problem Definition

Load balancing of temporary tasks is a online problem. In this problem, arriving tasks (or jobs) are to be assigned to processors, which are also called machines. In this entry, deterministic online load balancing of temporary tasks with unknown duration is discussed. The input sequence consists of departures and arrivals of tasks. If the sequence consists of arrivals only, the tasks are called permanent. Events happen one by one, so that the next event appears after the algorithm completed dealing with the previous event.

Clearly, the problem with temporary tasks is different from the problem with permanent tasks. One such difference is that for permanent tasks, the maximum load is always achieved in the end of the sequence. For temporary tasks this is not always the case. Moreover, the maximum load may be achieved at different times for different algorithms.

In the most general model, there are m machines $1, \ldots, m$. The information of an arriving job j is a vector p_j of length m, where p_j^i is the load or size of job j, if it is assigned to machine i. As stated above, each job is to be assigned to a machine before the next arrival or departure. The load of a machine i at time t is denoted by L_i^t and is the sum of the loads (on machine i) of jobs which are assigned to machine i, that arrived by time t and did not depart by this time. The goal is to minimize the maximum load of any machine over all times t. This machine model is known as *unrelated machines* (see [3] for a study of the load balancing problem of permanent tasks on unrelated machines). Many more specific models were defined. In the sequel a few such models are described.

For an algorithm $\mathcal{A}$, denote its cost by $\mathcal{A}$ as well. The cost of an optimal offline algorithm that knows the complete sequence of events in advance is denoted by OPT. Load balancing is studied in terms of the (absolute) competitive ratio. The competitive ratio of $\mathcal{A}$ is the infimum $\mathcal{R}$ such that for any input, $\mathcal{A} \leq \mathcal{R} \cdot \text{OPT}$. If the competitive ratio of an online algorithm is at most C it is also called C-competitive.

Uniformly related machines [3,12] are machines with speeds associated with them, thus machine i has speed s_i and the information that a job j needs to provide upon its arrival is just its size, or the load that it incurs on a unit speed machine, which is denoted by p_j. Then let $p_j^i = p_j/s_i$. If all speeds are equal, this results in *identical machines* [15].

Restricted assignment [8] is a model where each job may be run only on a subset of the machines. A job j is associated with running time which is the time to run it

on any of its permitted machines M_j. Thus if $i \in M_j$ then $p_j^i = p_j$ and otherwise $p_j^i = \infty$.

Key Results

The known results in all four models are surveyed below.

Identical Machines

Interestingly, the well known algorithm of Graham [15], LIST SCHEDULING, which is defined for identical machines, is valid for temporary tasks as well as permanent tasks. This algorithm greedily assigns a new job to the least loaded machine. The competitive ratio of this algorithm is $2 - 1/m$, which is best possible (see [5]). Note that the competitive ratio is the same as for permanent tasks, but for permanent tasks, it is possible to achieve a competitive ratio which does not tend to 2 for large m, see e. g. [11].

Uniformly Related Machines

The situation for uniformly related machines is not very different. In this case, the algorithms of Aspnes et al. [3] and of Berman et al. [12] cannot applied as they are, and some modifications are required. The algorithm of Azar et al. [7] has competitive ratio of at most 20 and it is based on the general method introduced in [3]. The algorithm of [3] keeps a guess value λ, which is an estimation of the cost of an optimal offline algorithm OPT. An invariant that must be kept is $\lambda \leq 2\text{OPT}$. At each step, a procedure is applied for some value of λ (which can be initialized as the load of the first job on the fastest machine). The procedure for a given value of λ is applied until it fails, and some job cannot be assigned while satisfying all conditions. The procedure is designed so that if it fails, then it must be the case that $\text{OPT} > \lambda$, the value of λ is doubled, and the procedure is re-invoked for the new value, ignoring all assignments that were done for small values of λ. This method is called doubling, and results in an algorithm with a competitive ratio which is at most four times the competitive ratio achieved by the procedure. The procedure for a given λ acts as follows. Let c be a target competitive ratio for the procedure. The machines are sorted according to speed. Each job is assigned to the first machine in the sorted order such that the job is assignable to it. A job j arriving at time t is assignable to machine i if $p_j/s_i \leq \lambda$ and $L_i^{t-1} + p_j/s_i \leq c\lambda$. It is shown in [7] that $c = 5$ allows the algorithm to succeed in the assignment of all jobs (i. e., to have at least one assignable machine for each job), as long as $\text{OPT} \leq \lambda$. Note that the constant c for permanent tasks used in [3] is 2. As for lower bounds, it is shown in [7] that the competitive ratio $\mathcal{R}$ of any algorithm satisfies $\mathcal{R} \geq 3 - o(1)$. The upper bound has been improved to $6 + 2\sqrt{5} \approx 10.47$ by Bar-Noy et al. [9].

Restricted Assignment

As for restricted assignment, temporary tasks make this model much more difficult than permanent tasks. The competitive ratio $O(\log m)$ which is achieved by a simple greedy algorithm (see [8]) does not hold in this case. In fact, the competitive ratio of this algorithm becomes $\Omega(m^{\frac{2}{3}})$ [4]. Moreover, in the same paper, a lower bound of $\Omega\sqrt{m}$ on the competitive ratio of any algorithm was shown. The construction was quite involved, however, Ma and Plotkin [16] gave a simplified construction which yields the same result.

The construction of [16] selects a value p which is the largest integer that satisfies $p + p^2 \leq m$. Clearly $p = \Theta(\sqrt{m})$. The lower bound uses two sets of machines, p machines which are called "the small group", and p^2 machines which are called "the large group". The construction consists of p^2 phases, each of which consists of p jobs and is dedicated to one machine in the large group. In phase i, job k of this phase can run either on the k-th machine of the small group, or the i-th machine of the large group. After this arrival, only one of these p jobs does not depart. An optimal offline algorithm assigns all jobs in each phase to the small group, except for the one job that will not depart. Thus when the construction is completed, it has one job on each machine of the large group. The maximum load ever achieved by OPT is 1. However, the algorithm does not know at each phase which job will not depart. If no job is assigned to the small group in phase i, then the load of machine i becomes p. Otherwise, a job that the algorithm assigns to the small group is chosen as the one that will not depart. In this way, after p phases, a total load of p^2 is accumulated on the small group, which means that at least one machine there has load p. This completed the construction.

An alternative algorithm called ROBIN HOOD was designed in [7]. This algorithm keeps a lower bound on OPT, which is the maximum between the following two functions. The first one is the maximum average machine load over time. The second is the maximum job size that has ever arrived. Denote this lower bound at time t (after t events have happened) by B^t. A machine i is called rich at time t if $L_i^t \geq \sqrt{m}B^t$. Otherwise it is called poor. The windfall time of a rich machine i at time t is the time t' such that i is poor at time $t' - 1$ and rich at times $t', \ldots, t$, i. e., the last time that machine i became rich. Clearly, machines can become poor due to an update of B^t or departure of jobs. A machine can become rich due to arrival of jobs that are assigned to it.

The algorithm assigns a job j to a poor machine in $M(j)$, if such a machine exists. Otherwise, j is assigned to the machine in $M(j)$ with the most recent windfall time. The analysis makes use of the fact that at most $\sqrt{m}$ machines can be rich simultaneously.

Note that for small values of m ($m \leq 5$), the competitive ratio of the greedy algorithm is still best possible, as shown in [1]. In this paper it was shown that these bounds are $(m+3)/2$ for $m = 3, 4, 5$. It is not difficult to see that for $m = 2$, the best bound is 2.

Unrelated Machines

The most extreme difference occurs for unrelated machines. Unlike the case of permanent tasks, where an upper bound of $O(\log m)$ can be achieved [3], it was shown in [2] that any algorithm has a competitive ratio of $\Omega(m/\log m)$. Note that a trivial algorithm, which assigns each job to the machine where it has a minimum load, has competitive ratio of at most m [3].

Applications

A similar model is known for the bin packing problem as well. In this problem, the sequence consists of arrivals and departures items of sizes in $(0, 1]$. The goal is to keep a partition of the currently existing items into subsets (bins), where the sum of items in each subset is at most 1. The cost is the maximum number of bins ever used simultaneously. This problem was studied in [13,14].

In [10], an hierarchical model was studied. This is a special case of restricted assignment where for each job j, $M(j)$ is a prefix of the machines. They showed that even for temporary tasks an algorithm of constant competitive ratio exists for this model.

In [6], which studied resource augmentation in load balancing, temporary tasks were considered as well. Resource augmentation is a type of analysis where the online algorithm is compared to an optimal offline algorithm which has less machines.

Open Problems

Small gaps still remain for both uniformly related machines, and for unrelated machines. For unrelated machines is could be interesting to find if there exists an algorithm of competitive ratio $o(m)$, or whether the simple algorithm stated above has optimal competitive ratio (up to a multiplicative factor).

Cross References

See ► List Scheduling for more information on identical machines and [15].

Recommended Reading

1. Armon, A., Azar, Y., Epstein, L., Regev, O.: On-line restricted assignment of temporary tasks with unknown durations. Inf. Process. Lett. **85**(2), 67–72 (2003)
2. Armon, A., Azar, Y., Epstein, L., Regev, O.: Temporary tasks assignment resolved. Algorithmica **36**(3), 295–314 (2003)
3. Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., Waarts, O.: On-line load balancing with applications to machine scheduling and virtual circuit routing. J. ACM **44**, 486–504 (1997)
4. Azar, Y., Broder, A.Z., Karlin, A.R.: On-line load balancing. Theor. Comput. Sci. **130**, 73–84 (1994)
5. Azar, Y., Epstein, L.: On-line load balancing of temporary tasks on identical machines. SIAM J. Discret. Math. **18**(2), 347–352 (2004)
6. Azar, Y., Epstein, L., van Stee, R.: Resource augmentation in load balancing. J. Sched. **3**(5), 249–258 (2000)
7. Azar, Y., Kalyanasundaram, B., Plotkin, S., Pruhs, K., Waarts, O.: On-line load balancing of temporary tasks. J. Algorithms **22**(1), 93–110 (1997)
8. Azar, Y., Naor, J., Rom, R.: The competitiveness of on-line assignments. J. Algorithms **18**, 221–237 (1995)
9. Bar-Noy, A., Freund, A., Naor, J.: New algorithms for related machines with temporary jobs. J. Sched. **3**(5), 259–272 (2000)
10. Bar-Noy, A., Freund, A., Naor, J.: On-line load balancing in a hierarchical server topology. SIAM J. Comput. **31**, 527–549 (2001)
11. Bartal, Y., Fiat, A., Karloff, H., Vohra, R.: New algorithms for an ancient scheduling problem. J. Comput. Syst. Sci. **51**(3), 359–366 (1995)
12. Berman, P., Charikar, M., Karpinski, M.: On-line load balancing for related machines. J. Algorithms **35**, 108–121 (2000)
13. Chan, W.-T., Wong, P.W.H., Yung, F.C.C.: On dynamic bin packing: an improved lower bound and resource augmentation analysis. In: Proc. of the 12th Annual International Conference on Computing and Combinatorics (COCOON2006), 2006, pp. 309–319
14. Coffman, E.G., Garey, M.R., Johnson, D.S.: Dynamic bin packing. SIAM J. Comput. **12**(2), 227–258 (1983)
15. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell Syst. Tech. J. **45**, 1563–1581 (1966)
16. Ma, Y., Plotkin, S.: Improved lower bounds for load balancing of tasks with unknown duration. Inf. Process. Lett. **62**, 31–34 (1997)

Local Alignment (with Affine Gap Weights)

1986; Altschul, Erickson

Stephen F. Altschul[1,2], Bruce W. Erickson[1], Henry Leung[2]

[1] The Rockefeller University, New York, NY, USA

[2] Department of Applied Mathematics, MIT, Cambridge, MA, USA

Keywords and Synonyms

Pairwise local alignment with affine gap weight

Problem Definition

The pairwise local alignment problem is concerned with identification of a pair of similar substrings from two molecular sequences. This problem has been studied in computer science for four decades. However, most problem models were generally not biologically satisfying or interpretable before 1974. In 1974, Sellers developed a metric measure of the similarity between molecular sequences. Waterman et al. (1976) generalized this metric to include deletions and insertions of arbitrary length which represent the minimum number of mutational events required to convert one sequence into another.

Given two sequences S and T, a pairwise alignment is a way of inserting space characters '_' in S and T to form sequences S' and T' respectively with the same length. There can be different alignments of two sequences. The score of an alignment is measured by a scoring metric $\delta(x, y)$. At each position i where both x and y are not spaces, the similarity between $S'[i]$ and $T'[i]$ is measured by $\delta(S'[i], T'[j])$. Usually, $\delta(x, y)$ is positive when x and y are the same and negative when x and y are different. For positions with consecutive space characters, the alignment scores of the space characters are not considered independently; this is because inserting or deleting a long region in molecular sequences is more likely to occur than inserting or deleting several short regions. Smith and Waterman use an affine gap penalty to model the similarity at positions with space characters. They define a consecutive substring with spaces in S' or T' be a gap. For each length l gap, they give a linear penalty $W_k = W_s + l \times W_p$ for some predefined positive constants W_s and W_p. The score of an alignment is the sum of the score at each position i minus the penalties of each gap. For example, the alignment score of the following alignment is $\delta(G, G) + \delta(C, C) + \delta(C, C) + \delta(U, C) + \delta(G, G) - (W_s + 2 \times W_p)$.

S: $GCCAUUG$

T: $GCC\_\_CG$

The optimal global alignment of sequences S and T is the alignment of S and T with the maximum alignment score.

Sometimes we want to know whether sequences S and T contain similar substrings instead of whether S and T are similar. In this case, they solve the pairwise local alignment problem, which wants to find a substring U in S and another substring V in T such that the global alignment score of U and V is maximized.

Pairwise Local Alignment Problem

Input: Two sequences $S[1..n]$ and $T[1..m]$.
Output: A substring U in S and a substring V in T such that the optimal global alignment of U and V is maximized.

Key Results

The pairwise local alignment problem can be solved in O(mn) time and O(mn) space by dynamic programming. The algorithm needs to fill in the 4 $m \times n$ tables H, H_N, H_S and H_T, where each entry takes constant time. The individual meanings of these 4 tables are as follows.

$H(i, j)$: maximum score of the global alignment of U and V over all suffixes U in $S[1..i]$ and all suffixes V in $T[1..j]$.

$H_N(i, j)$: maximum score of the global alignment of U and V over all suffixes U in $S[1..i]$ and all suffixes V in $T[1..j]$, with the restriction that $S[i]$ and $T[j]$ must be aligned.

$H_S(i, j)$: maximum score of the global alignment of U and V over all suffixes U in $S[1..i]$ and all suffixes V in $T[1..j]$, with $S[j]$ aligned with a space character.

$H_T(i, j)$: maximum score of the global alignment of U and V over all suffixes U in $S[1..i]$ and all suffixes V in $T[1..j]$, with $T[j]$ aligned with a space character.

The optimal local alignment score of S and T will be $\max\{H(i, j)\}$ and the local alignment of S and T can be found by tracking back table H.

In the tables, each entry can be filled in by the following recursion in constant time.

Basic Step:

$$
\begin{aligned}
H(i, 0) = H(0, j) &= 0\,, && 0 \leqslant i \leqslant n\,,\ 0 \leqslant i \leqslant m \\
H_N(i, 0) = H_N(0, j) &= -\infty\,, && 0 \leqslant i \leqslant n\,,\ 0 \leqslant i \leqslant m \\
H_s(i, 0) = H_T(0, j) &= W_s + W_p\,, && 0 \leqslant i \leqslant n\,,\ 0 \leqslant i \leqslant m \\
H_s(0, j) = H_T(i, 0) &= -\infty\,, && 0 \leqslant i \leqslant n\,,\ 0 \leqslant i \leqslant m
\end{aligned}
$$

Recursion Step:

$$
\begin{aligned}
H(i, j) &= \max\{H_N(i, j), H_s(i, j), H_T(i, j), 0\}\,, \\
&\qquad 1 \leqslant i \leqslant n\,,\ 1 \leqslant i \leqslant m \\
H_N(i, j) &= H(i-1, j-1) + \delta(S[i], T[j])\,, \\
&\qquad 1 \leqslant i \leqslant n\,,\ 1 \leqslant i \leqslant m
\end{aligned}
$$

$$H_s(i,j) = \max\{H(i-1,j) - (W_s + W_p), H_S(i-1,j) - W_p\}, \quad 1 \leqslant i \leqslant n, \ 1 \leqslant i \leqslant m$$

$$H_T(i,j) = \max\{H(i,j-1) - (W_s + W_p), H_T(i,j-1) - W_p\}, \quad 1 \leqslant i \leqslant n, \ 1 \leqslant i \leqslant m$$

Applications

Local alignment with affine gap penalty can be used for protein classification, phylogenetic footprinting and identification of functional sequence elements.

URL to Code

http://bioweb.pasteur.fr/seqanal/interfaces/water.html

Cross References

- ▶ Efficient Methods for Multiple Sequence Alignment with Guaranteed Error Bounds
- ▶ Local Alignment (with Concave Gap Weights)

Recommended Reading

1. Allgower, E.L., Schmidt, P.H.: An Algorithm for Piecewise-Linear Approximation of an Implicitly Defined Manifold. SIAM J. Num. Anal. **22**, 322–346 (1985)
2. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic Local Alignment Search Tool. J. Mol. Biol. **215**, 403–410 (1990)
3. Chao, K.M., Miller, W.: Linear-space algorithms that build local alignments from fragments. Algorithmica **13**, 106–134 (1995)
4. Myers, E.W., Miller, W.: Optimal Alignments in Linear Space. Bioinformatics **4**, 11–17 (1988)
5. Gusfield, D.: Algorithms on Strings, Trees and Sequences. Cambridge University Press, Cambridge (1999). ISBN 052158519
6. Ma, B., Tromp, J., Li, M.: PatternHunter: Faster and More Sensitive Homology Search. Bioinformatics **18**, 440–445 (2002)
7. Needleman, S.B., Wunsch, C.D.: A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. J. Mol. Biol. **48**, 443–453 (1970)
8. Pearson, W.R., Lipman, D.J.: Improved Tools for Biological Sequence Comparison. Proc. Natl. Acad. Sci. USA **85**, 2444–2448 (1988)
9. Sellers, P.H.: On the Theory and Computation of Evolutionary Distances. SIAM J. Appl. Math. **26**, 787–793 (1974)
10. Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. **147**, 195–197 (1981)

Local Alignment (with Concave Gap Weights)

1988; Miller, Myers

S. M. Yiu
Department of Computer Science, The University of Hong Kong, Hong Kong, China

Keywords and Synonyms

Sequence alignment; Pairwise local alignment

Problem Definition

This work of Miller and Myers [9] deals with the problem of pairwise sequence alignment in which the distance measure is based on the gap penalty model. They proposed an efficient algorithm to solve the problem when the gap penalty is a concave function of the gap length.

Let X and Y be two strings (sequences) of alphabet Σ. The pairwise alignment $\mathcal{A}$ of X and Y maps X, Y into strings X', Y' that may contain spaces (not in Σ) such that (1) $|X'| = |Y'| = \ell$; (2) removing spaces from X' and Y' returns X and Y, respectively; and (3) for any $1 \leq i \leq \ell$, $X'[i]$ and $Y'[i]$ cannot be both spaces where $X'[i]$ denotes the ith character in X'.

To evaluate the quality of an alignment, there are many different measures proposed (e. g. edit distance, scoring matrix [11]). In this work, they consider the *gap penalty* model.

A *gap* in an alignment $\mathcal{A}$ of X and Y is a maximal substring of contiguous spaces in either X' or Y'. There are gaps and aligned characters (both $X'[i]$ and $Y'[i]$ are not space) in an alignment. The score for a pair of aligned characters is based on a distance function $\delta(a,b)$ where $a, b \in \Sigma$. Usually δ is a metric, but this assumption is not required in this work. The penalty of a gap of length k is based on a non-negative function $W(k)$. The score of an alignment is the sum of the scores of all aligned characters and gaps. An alignment is *optimal* if its score is the minimum possible.

The penalty function $W(k)$ is *concave* if $\bigtriangleup W(k) \geq \bigtriangleup W(k+1)$ for all $k \geq 1$, where $\bigtriangleup W(k) = W(k+1) - W(k)$.

The penalty function $W(k)$ is *affine* if $W(k) = a + bk$ where a, b are constants. Affine function is a special case of concave function. The problem for affine gap penalty has been considered in [1,6].

The penalty function $W(k)$ is a *P-piece affine curve* if the domain of W can be partitioned into P intervals, $(\tau_1 = 1, \chi_1), (\tau_2, \chi_2), \ldots, (\tau_p, \chi_p = \infty)$, where $\tau_i = \chi_{i-1} + 1$ for all $1 < i \leq p$, such that for each interval, the values of W follow an affine function. More precisely, for any $k \in (\tau_i, \chi_i)$, $W(k) = a_i + b_i k$ for some constants a_i, b_i.

Problem

INPUT: Two strings X and Y, the scoring function δ, and the gap penalty function $W(k)$.
OUTPUT: An optimal alignment of X and Y.

Key Results

Theorem 1 *If $W(k)$ is concave, they provide an algorithm for computing an optimal alignment that runs in $O(n^2 \log n)$ time where n is the length of each string and uses $O(n)$ expected space.*

Corollary 1 *If $W(k)$ is an affine function, the same algorithm runs in $O(n^2)$ time.*

Theorem 2 *For some special types of gap penalty functions, the algorithm can be modified to run faster.*

- *If $W(k)$ is a P-piece affine curve, the algorithm can be modified to run in $O(n^2 \log P)$ time.*
- *For logarithmic gap penalty function, $W(k) = a + b \log k$, the algorithm can be modified to run in $O(n^2)$ time.*
- *If $W(k)$ is a concave function when $k > K$, the algorithm can be modified to run in $O(K + n^2 \log n)$ time.*

Applications

Pairwise sequence alignment is a fundamental problem in computational biology. Sequence similarity usually implies functional and structural similarity. So, pairwise alignment can be used to check whether two given sequences have similar functions or structures and to predict functions of newly identified DNA sequence. One can refer to Gusfield's book for some examples on the importance of sequence alignment (pp. 212–214 of [7]).

The alignment problem can be further divided into the *global* alignment problem and the *local* alignment problem. The problem defined here is the global alignment problem in which the whole input strings are required to align with each other. On the other hand, for local alignment, the main interest lies in identifying a substring from each of the input strings such that the alignment score of the two substrings is the minimum among all possible substrings. Local alignment is useful in aligning sequences that are not similar, but contain a region that are highly conserved (similar). Usually this region is a functional part (domain) of the sequences. Local alignment is particularly useful in comparing proteins. Proteins in the same family from different species usually have some functional domains that are highly conserved while the other parts are not similar at all. Examples are the homeobox genes [10] for which the protein sequences are quite different in each species except the functional domain *homeodomain*.

Conceptually, the alignment score is used to capture the evolutionary distance between the two given sequences. Since a gap of more than one space can be created by a single mutational event, so considering a gap of length k as a unit instead of k different point mutation may be more appropriate in some cases. However, which gap penalty function should be used is a difficult question to answer and sometimes depend on the actual applications. Most applications, such as BLAST, uses the affine gap penalty which is still the dominate model in practice. On the other hand, Benner et al. [2] and Gu and Li [13] suggested to use the logarithmic gap penalty in some cases. Whether using a concave gap penalty function in general is meaningful is still an open issue.

Open Problem

Note that the results of this paper have been independently obtained by Galil and Giancarlo [5] and for affine gap penalty, Gotoh [6] also gave an $O(n^2)$ algorithm for solving the alignment problem. In [4], Eppstein gave a faster algorithm that runs in $O(n^2)$ time for solving the same sequence alignment problem with concave gap penalty function. Whether a subquadratic algorithm exists for solving this problem remains open. As a remark, subquadratic algorithms do exist for solving the sequence alignment problem if the measure is not based on the gap penalty model, but is computed as $\sum_{i=1}^{\ell} \delta(X1'[i], Y'[i])$ based only on a scoring function $\delta(a, b)$ where $a, b \in \Sigma \cup \{\_\}$ where '_' represents the space [3,8].

Experimental Results

They have performed some experiments to compare their algorithm with Waterman's $O(n^3)$ algorithm [12] on a number of different concave gap penalty functions. Artificial sequences are generated for the experiments. Results from their experiments lead to their conjectures that Waterman's method runs in $O(n^3)$ time when the two given strings are very similar or the score for mismatch characters is small and their algorithm runs in $O(n^2)$ time if the range of the function $W(k)$ is not functionally dependent on n.

Cross References

► Local Alignment (with Affine Gap Weights)

Recommended Reading

1. Altschul, S.F., Erickson, B.W.: Optimal sequence alignment using affine gap costs. Bull. Math. Biol. **48**, 603–616 (1986)
2. Benner, S.A., Cohen, M.A., Gonnet, G.H.: Empirical and structural models for insertions and deletions in the divergent evolution of proteins. J. Mol. Biol. **229**, 1065–1082 (1993)
3. Crochemore, M., Landau, G.M., Ziv-Ukelson, M.: A subquadratic sequence alignment algorithm for unrestricted scoring matrices. SIAM J. Comput. **32**(6), 1654–1673 (2003)

4. Eppstein, D.: Sequence comparison with mixed convex and concave costs. J. Algorithms **11**(1), 85–101 (1990)
5. Galil, Z., Giancarlo, R.: Speeding up dynamic programming with applications to molecular biology. Theor. Comput. Sci. **64**, 107–118 (1989)
6. Gotoh, O.: An improved algorithm for matching biological sequences. J. Mol. Biol. **162**, 705–708 (1982)
7. Gusfield, D.: Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge (1997)
8. Masek, W.J., Paterson, M.S.: A fater algorithm for computing string edit distances. J. Comput. Syst. Sci. **20**, 18–31 (1980)
9. Miller, W., Myers, E.W.: Sequence comparison with concave weighting functions. Bull. Math. Biol. **50**(2), 97–120 (1988)
10. De Roberts, E., Oliver, G., Wright, C.: Homeobox genes and the vertibrate body plan, pp. 46–52. Scientific American (1990)
11. Sankoff, D., Kruskal, J.B.: Time Warps, Strings Edits, and Macromolecules: The Theory and Practice of Sequence Comparison. Addison-Wesley (1983)
12. Waterman, M.S.: Efficient sequence alignment algorithms. J. Theor. Biol. **108**, 333–337 (1984)
13. Li, W.-H., Gu, X.: The size distribution of insertions and deletions in human and rodent pseudogenes suggests the logarithmic gap penalty for sequence alignment. J. Mol. Evol. **40**, 464–473 (1995)

Local Approximation of Covering and Packing Problems

2003–2006; Kuhn, Moscibroda, Nieberg, Wattenhofer

Fabian Kuhn
Department of Computer Science, ETH Zurich, Zurich, Switzerland

Synomyms

Distributed approximation of covering and packing problems

Problem Definition

A *local algorithm* is a distributed algorithm on a network with a running time which is independent or almost independent of the network's size or diameter. Usually, a distributed algorithm is called local if its time complexity is at most polylogarithmic in the size n of the network. Because the time needed to send information from one node of a network to another is at least proportional to the distance between the two nodes, in such an algorithm, each node's computation is based on information from nodes in a close vicinity only. Although all computations are based on local information, the network as a whole typically still has to achieve a global goal. Having local algorithms is inevitable to obtain time-efficient distributed protocols for large-scale and dynamic networks such as peer-to-peer networks or wireless ad hoc and sensor networks.

In [2,6,7], Kuhn, Moscibroda, and Wattenhofer describe upper and lower bounds on the possible trade-off between locality (time complexity) of distributed algorithms and the quality (approximation ratio) of the achievable solution for an important class of problems called covering and packing problems. Interesting covering and packing problems in the context of networks include minimum dominating set, minimum vertex cover, maximum matching, as well as certain flow maximization problems. All the results given in [2,6,7] hold for general network topologies. Interestingly, it is shown by Kuhn, Moscibroda, Nieberg, and Wattenhofer in [3,4,5] that covering and packing problems can be solved much more efficiently when assuming that the network topology has special properties which seem realistic for wireless networks.

Distributed Computation Model

In [2,3,4,5,6,7], the network is modeled as an undirected and except for [5] unweighted graph $G = (V, E)$. Two nodes $u, v \in V$ of the network are connected by an edge $(u, v) \in E$ whenever there is a direct bidirectional communication channel connecting u and v. In the following, the number of nodes and the maximal degree of G are denoted by $n = |V|$ and by Δ.

For simplicity, communication is assumed to be synchronous. That is, all nodes start an algorithm simultaneously and time is divided into rounds. In each round, every node can send an arbitrary message to each of its neighbors and perform some local computation based on the information collected in previous rounds. The *time complexity* of a synchronous distributed algorithm is the number of rounds until all nodes terminate.

Local distributed algorithms in the described synchronous model have first been considered in [8] and [9]. As an introduction to the above and similar distributed computation models, it is also recommended to read [11].

Distributed Covering and Packing Problems

A fractional covering problem (P) and its dual fractional packing problem (D), are linear programs (LPs) of the canonical forms

$$\begin{array}{lll} \min & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} & \\ \text{s.t.} & A \cdot \boldsymbol{x} \geq \boldsymbol{b} & \text{(P)} \\ & \boldsymbol{x} \geq \boldsymbol{0} & \end{array} \qquad \begin{array}{lll} \max & \boldsymbol{b}^{\mathrm{T}}\boldsymbol{y} & \\ \text{s.t.} & A^{\mathrm{T}} \cdot \boldsymbol{y} \leq \boldsymbol{c} & \text{(D)} \\ & \boldsymbol{y} \geq \boldsymbol{0} & \end{array}$$

where all a_{ij}, b_i, and c_i are non-negative. In a distributed context, finding a small (weighted) dominating set or

a small (weighted) vertex cover of the network graph are the most important covering problems. A dominating set of a graph G is a subset S of its nodes such that all nodes of G either are in S or have a neighbor in S. The dominating set problem can be formulated as covering integer LP by setting A to be the adjacency matrix with 1s in the diagonal, by setting $\boldsymbol{b}$ to be a vector with all 1s and if $\boldsymbol{c}$ is the weight vector. A vertex cover is a subset of the nodes such that all edges are covered. Packing problems occur in a broad range of resource allocation problems. As an example, in [1] and [10], the problem of assigning flows to a given fixed set of paths is described. Another common packing problem is (weighted) maximum matching, the problem of finding a largest possible set of pairwise non-adjacent edges.

While computing a dominating set, vertex cover, or matching of the network graph are inherently distributed tasks, general covering and packing LPs have no immediate distributed meaning. To obtain a distributed version of these LPs, two dual LPs (P) and (D) are mapped to a bipartite network as follows. For each primal variable x_i and for each dual variable y_j, there are nodes v_i^p and v_j^d, respectively. There is an edge between two nodes v_i^p and v_j^d whenever $a_{ji} \neq 0$, i. e., there is an edge if the ith variable of an LP occurs in its jth inequality.

In most real-world examples of distributed covering and packing problems, the network graph is of course not equal to the described bipartite graph. However, it is usually straightforward to simulate an algorithm which is designed for the above bipartite network on the actual network graph without affecting time and message complexities.

Bounded Independence Graphs

In [3,4,5], local approximation algorithms for covering and packing problems for graphs occuring in the context of wireless ad hoc and sensor networks are studied. Because of scale, dynamism and the scarcity of resources, these networks are a particular interesting area to apply local distributed algorithms.

Wireless networks are often modeled as *unit disk graphs* (UDGs): Nodes are assumed to be in a two-dimensional Euclidean plane and two nodes are connected by an edge iff their distance is at most 1. This certainly captures the inherent geometric nature of wireless networks. However, unit disk graphs seem much too restrictive to accurately model real wireless networks. In [3,4,5], Kuhn et. al. therefore consider two generalizations of the unit disk graph model, *bounded independent graphs* (BIGs) and *unit ball graphs* (UBGs). A BIG is a graph where all local independent sets are of bounded size. In particular, it is assumed that there is a function $I(r)$ which upper bounds the size of the largest independent set of every r-neighborhood in the graph. Note that the value of $I(r)$ is independent of n, the size of the network. If $I(r)$ is a polynomial in r, a BIG is said to be polynomially bounded. UDGs are BIGs with $I(r) \in O(r^2)$. UBGs are a natural generalization of UDGs. Given some underlying metric space (V, d) two nodes $u, v \in V$ are connected by an edge iff $d(u, v) \leq 1$. If the metric space (V, d) has constant doubling dimension[1], a UBG is a polynomially bounded BIG.

Key Results

The first algorithms to solve general distributed covering and packing LPs appear in [1,10]. In [1], it is shown that it is possible to find a solution which is within a factor of $1 + \varepsilon$ of the optimum in $O(\log^3(\rho n)/\varepsilon^3)$ rounds where ρ is the ratio between the largest and the smallest non-zero coefficient of the LPs. The result of [1] is improved and generalized in [6,7] where the following result is proven:

Theorem 1 *In k rounds, (P) and (D) can be approximated by a factor of $(\rho\Delta)^{O(1/\sqrt{k})}$ using messages of size at most $O(\log(\rho\Delta))$. An $(1 + \varepsilon)$-approximation can be found in time $O(\log^2(\rho\Delta)/\varepsilon^4)$.*

The algorithm underlying Theorem 1 needs only small messages of size $O(\log(\rho\Delta))$ and extremely simple and efficient local computations. If larger messages and more complicated (but still polynomial) local computations are allowed, it is possible to improve the result of Theorem 1:

Theorem 2 *In k rounds, LPs of the form (P) or (D) can be approximated by a factor of $O(n^{O(1/k)})$. This implies that a constant approximation can be found in time $O(\log n)$.*

Theorems 1 and 2 only give bounds on the quality of distributed solutions of covering and packing LPs. However, many of the practically relevant problems are integer versions of covering and packing LPs. Combined with simple randomized rounding schemes, the following upper bounds for dominating set, vertex cover, and matching are proven in [6,7]:

Theorem 3 *Let Δ be the maximal degree of the given network graph. In k rounds, minimum dominating set can be approximated by a factor of $O(\Delta^{O(1/\sqrt{k})} \cdot \log \Delta)$ in expectation by using messages of size $O(\Delta)$. Without bound on the message size, an expected approximation ratio of*

[1] The doubling dimension of a metric space is the logarithm of the maximal number of balls needed to cover a ball $B_r(x)$ in the metric space with balls $B_{r/2}(y)$ of half the radius.

$O(n^{O(1/k)} \cdot \log \Delta)$ can be achieved. Minimum vertex cover and maximum matching can both be approximated by a factor of $O(\Delta^{1/k})$ in k rounds.

In [2,7], it is shown that the upper bounds on the trade-offs between time complexity and approximation ratio given by Theorems 1–3 are almost optimal:

Theorem 4 *In k rounds, it is not possible to approximate minimum vertex cover better than by factors of $\Omega(\Delta^{1/k}/k)$ and $\Omega(n^{\Omega(1/k^2)}/k)$. This implies time lower bounds of $\Omega(\log \Delta/\log\log \Delta)$ and $\Omega(\sqrt{\log n/\log\log n})$ for constant or even poly-logarithmic approximation ratios. The same bounds hold for minimum dominating set, for maximum matching, as well as for the underlying LPs.*

While Theorem 4 shows that the results given by Theorems 1–3 are close to optimal for worst-case network topologies, the problems might be much simpler if restricted to networks which actually occur in reality. In fact, it is shown in [3,4,5] that the above results can indeed be improved if the network graph is assumed to be a BIG or a UBG with constant doubling dimension. In [5], the following result for UBGs is proven:

Theorem 5 *Assume that the network graph $G = (V, E)$ is a UBG with underlying metric (V, d). If (V, d) has constant doubling dimension and if all nodes know the distances to their neighbors in G up to a constant factor, it is possible to find constant approximations for minimum dominating set, minimum vertex cover, maximum matching, as well as for general LPs of the forms (P) and (D) in $O(\log^* n)$ rounds*[2].

While the algorithms underlying the results of Theorems 1 and 2 for solving covering and packing LPs are deterministic or straight-forward to be derandomized, all known efficient algorithms to solve minimum dominating set and more complicated integer covering and packing problems are randomized. Whether there are good deterministic local algorithms for dominating set and related problems is a long-standing open question . In [3], it is shown that if the network is a BIG, efficient deterministic distributed algorithms exist:

Theorem 6 *On a BIG it is possible to find constant approximations for minimum dominating set, minimum vertex cover, maximum matching, as well as for LPs of the forms (P) and (D) deterministically in $O(\log \Delta \cdot \log^* n)$ rounds.*

In [4], it is shown that on polynomially bounded BIGs, one can even go one step further and efficiently find an arbitrarily good approximation by a distributed algorithm:

Theorem 7 *On a polynomially bounded BIG, there is a local approximation scheme which computes a $(1+\varepsilon)$-approximation for minimum dominating set in time $O(\log \Delta \log^*(n)/\varepsilon + 1/\varepsilon^{O(1)})$. If the network graph is a UBG with constant doubling dimension and nodes know the distances to their neighbors, a $(1+\varepsilon)$-approximation can be computed in $O(\log^*(n)/\varepsilon + 1/\varepsilon^{O(1)})$ rounds.*

Applications

The most important application environments for local algorithms are large-scale decentralized systems such as wireless ad hoc and sensor networks or peer-to-peer networks. On such networks, only local algorithms lead to scalable systems. Local algorithms are particularly well-suited if the network is dynamic and computations have to be repeated frequently.

A particular application of the minimum dominating set problem is the task of clustering the nodes of wireless ad hoc or sensor networks. Assigning each node to an adjacent node in a dominating set induces a simple clustering of the nodes. If the nodes of the dominating set (i. e., the cluster centers) are connected with each other by using additional nodes, the resulting structure can be used as a backbone for routing.

Open Problems

There are a number of open problems related to the distributed approximation of covering and packing problems in particular and to distributed approximation algorithms in general. The most obvious open problem certainly is to close the gaps between the upper bounds of Theorems 1, 2, and 3 and the lower bounds of Theorem 4. It would also be interesting to see how well other optimization problems can be approximated in a distributed manner. In particular, the distributed complexity of more general classes of linear programs remains completely open. A very intriguing unsolved problem is to determine to what extent randomization is needed to obtain time-efficient distributed algorithms. Currently, the best determinic algorithms for finding a dominating set of reasonable size and for many other problems take time $2^{O(\sqrt{\log n})}$ whereas the time complexity of the best randomized algorithms usually is at most polylogarithmic in the number of nodes.

Cross References

- ► Fractional Packing and Covering Problems
- ► Maximum Matching
- ► Randomized Rounding

[2] The log-star function $\log^* n$ is an extremely slowly increasing function which gives the number of times the logarithm has to be taken to obtain a number smaller than 1.

Recommended Reading

1. Bartal, Y., Byers, J.W., Raz, D.: Global optimization using local information with applications to flow control. In: Proc. of the 38th IEEE Symposium on the Foundations of Computer Science (FOCS), pp. 303–312 (1997)
2. Kuhn, F., Moscibroda, T., Wattenhofer, R.: What cannot be computed locally! In: Proc. of the 23rd ACM Symp. on Principles of Distributed Computing (PODC), pp. 300–309 (2004)
3. Kuhn, F., Moscibroda, T., Nieberg, T., Wattenhofer, R.: Fast deterministic distributed maximal independent set computation on growth-bounded graphs. In: Proc. of th 19th Int. Conference on Distributed Computing (DISC), pp. 273–287 (2005)
4. Kuhn, F., Moscibroda, T., Nieberg, T., Wattenhofer, R.: Local approximation schemes for ad hoc and sensor networks. In: Proc. of the 3rd Joint Workshop on Foundations of Mobile Computing (DIALM-POMC), pp. 97–103 (2005)
5. Kuhn, F., Moscibroda, T., Wattenhofer, R.: On the locality of bounded growth. In: Proc. of the 24th ACM Symposium on Principles of Distributed Computing (PODC), pp. 60–68 (2005)
6. Kuhn, F., Wattenhofer, R.: Constant-time distributed dominating set approximation. Distrib. Comput. **17**(4), 303–310 (2005)
7. Kuhn, F., Moscibroda, T., Wattenhofer, R.: The price of being near-sighted. In: Proc. of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 980–989 (2006)
8. Linial, N.: Locality in distributed graph algorithms. SIAM J. Comput. **21**(1), 193–201 (1992)
9. Naor, M., Stockmeyer, L.: What can be computed locally? In: Proc. of the 25th Annual ACM Symposium on Theory of Computing (STOC), pp. 184–193 (1993)
10. Papadimitriou, C., Yannakakis, M.: Linear programming without the matrix. In: Proc. of the 25th ACM Symposium on Theory of Computing (STOC), pp. 121–129 (1993)
11. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM (2000)

Local Computation in Unstructured Radio Networks

2005; Moscibroda, Wattenhofer

THOMAS MOSCIBRODA
Systems & Networking Research Group, Microsoft Research, Redmond, WA, USA

Keywords and Synonyms

Maximal independent sets in radio networks; Coloring unstructured radio networks

Problem Definition

In many ways, familiar distributed computing communication models such as the *message passing model* do not describe the harsh conditions faced in wireless ad hoc and sensor networks closely enough. Ad hoc and sensor networks are multi-hop radio networks and hence, messages being transmitted may interfere with concurrent transmissions leading to collisions and packet losses. Furthermore, the fact that all nodes share the same wireless communication medium leads to an inherent broadcast nature of communication. A message sent by a node can be received by all nodes in its transmission range. These aspects of communication are modeled by the *radio network model*, e. g. [2].

Definition 1 (Radio Network Model) In the radio network model, the wireless network is modeled as a graph $G = (V, E)$. In every time-slot, a node $u \in V$ can either send or not send a message. A node v, $(u, v) \in E$, receives the message if and only *exactly one* of its neighbors has sent a message in this time-slot.

While communication primitives such as broadcast, wake-up, or gossiping, have been widely studied in the literature on radio networks (e. g., [1,2,8]), less is known about the computation of *local network coordination structures* such as clusterings or colorings. The most basic notion of a clustering in wireless networks boils down to the graph-theoretic notion of a dominating set.

Definition 2 (Minimum Dominating Set (MDS)) Given a graph $G = (V, E)$. A dominating set is a subset $S \subseteq V$ such that every node is either in S or has at least one neighbor in S. The minimum dominating set problem asks for a dominating set S of minimum cardinality.

A dominating set $S \subseteq V$ in which no two neighboring nodes are in S is a *maximal independent set (MIS)*. The distributed complexity of computing a MIS in the message passing model has been of fundamental interest to the distributed computing community for over two decades (e. g., [11,12,13]), but much less is known about the problem's complexity in radio network models.

Definition 3 (Maximal Independent Set (MIS)) Given a graph $G = (V, E)$. An independent set is a subset of pairwise non-adjacent nodes in G. A maximal independent set in G is an independent set $S \subseteq V$ such that for every node $u \notin S$, there is a node $v \in \Gamma(u)$ in S.

Another important primitive in wireless networks is the *vertex coloring* problem, because associating different colors with different time slots in a time-division multiple access (TDMA) scheme; a correct coloring corresponds to a medium access control (MAC) layer without *direct interference*, that is, no two neighboring nodes send at the same time.

Definition 4 (Minimum Vertex Coloring) Given a graph $G = (V, E)$. A correct vertex coloring for G is an assignment of a color $c(v)$ to each node $v \in V$, such that

$c(u) \neq c(v)$ any two adjacent nodes $(u, v) \in E$. A minimum vertex coloring is a correct coloring that minimizes the number of used colors.

In order to capture the especially harsh characteristics of wireless multi-hop networks immediately after their deployment, the unstructured radio network model makes additional assumptions. In particular, a new notion of asynchronous wake-up is considered, because, in a wireless, multi-hop environment, it is realistic to assume that some nodes join the network (e. g. become deployed, or switched on) later than others. Notice that this is different from the notion of asynchronous wake-up defined and studied in [8] and subsequent work, in which nodes are assumed to be "woken up" by incoming messages.

Definition 5 (Unstructured Radio Network Model) In the *unstructured radio network model*, the wireless network is modeled as a unit disk graph (UDG) $G = (V, E)$. In every time-slot, a node $u \in V$ can either send or not send a message. A node v, $(u, v) \in E$, receives the message if and only *exactly one* of its neighbors has sent a message in this time-slot. Additionally, the following assumptions are made:

- *Asynchronous wake-up:* New nodes can wake up/join in *asynchronously* at any time. Before waking-up, nodes do neither receive nor send any messages.
- *No global clock:* Nodes only have access to a local clock that starts increasing after wake-up.
- *No collision detection:* Nodes cannot distinguish between the event of a collision and no message being sent. Moreover, a sending node does not know how many (if any at all!) neighbors have received its transmission correctly.
- *Minimal global knowledge:* At the time of their wake-up, nodes have no information about their neighbors in the network and they do not whether some neighbors are already awake, executing the algorithm. However, nodes know an upper bound for the maximum number of nodes $n = |V|$.

The measure that captures the efficiency of an algorithm defined in the unstructured radio network model is its *time-complexity*. Since every node can wake up at a different time, the time-complexity of an algorithm is defined as the maximum number of time-slots between a node's wake-up and its final, irrevocable decision.

Definition 6 (Time Complexity) The *running time* T_v of a node $v \in V$ is defined as the number of time slots between v's *waking up* and the time v makes an irrevocable *final decision* on the outcome of its protocol (e. g. whether or not it joins the dominating set in a clustering algorithm, or which color to take in a coloring algorithm, etc.). The *time complexity* $T(\mathcal{Q})$ of algorithm $\mathcal{Q}$ is defined as the maximum running time over all nodes in the network, i. e., $T(\mathcal{Q}) := \max_{v \in V} T_v$.

Key Results

Naturally, algorithms for such uninitialized, chaotic networks have a different flavor compared to "traditional" algorithms that operate on a given network graph that is static and well-known to all nodes. Hence, the algorithmic difficulty of the following algorithms partly stems from the fact that since nodes wake up asynchronously and do not have access to a global clock, the different phases of the algorithm may be arbitrarily intertwined or shifted in time. Hence, while some nodes may already be in an advanced stage of the algorithm, there may be nodes that have either just woken up, or that are still in early stage. It was proven in [9] that even in single-hop networks (G is the complete graph), no efficient algorithms exist if nodes have no knowledge on n.

Theorem 1 *If nodes have no knowledge of n, every (possibly randomized) algorithm requires up to $\Omega(n/\log n)$ time slots before at least one node can send a message in single-hop networks.*

In single-hop networks, and if n is globally known, [8] presented a randomized algorithm that selects a unique leader in time $O(n \log n)$, with high probability. This result has subsequently been improved to $O(\log^2 n)$ by Jurdziński and Stachowiak [9]. The generalized wake-up problem in multi-hop radio network was first studied in [4].

The complexity of local network structures such as clusterings or colorings in unstructured multi-hop radio networks was first studied in [10]: A good approximation to the minimum dominating set problem can be computed in polylogarithmic time.

Theorem 2 *In the unstructured radio network model, an expected $O(1)$-approximation to the dominating set problem can be computed in expected time $O(\log^2 n)$. That is, every node decides whether to join the dominating set within $O(\log^2 n)$ time slots after its wake-up.*

In a subsequent paper [18], it has been shown that the running time of $O(\log^2 n)$ is sufficient even for computing the more sophisticated MIS structure. This result is asymptotically optimal because—improving on a previously known bound of $\Omega(\log^2 n/\log\log n)$ [9]—, a corresponding lower bound of $\Omega(\log^2 n)$ has been proven in [6].

Theorem 3 *With high probability, a maximal independent set (MIS) can be computed in expected time $O(\log^2 n)$*

in the unstructured radio network model. This is asymptotically optimal.

It is interesting to compare this achievable upper bound on the harsh unstructured radio network model with the best known time lower bounds in message passing models: $\Omega(\log^* n)$ in unit disk graphs [12] and $\Omega(\sqrt{\log n / \log\log n})$ in general graphs [11]. Also, a time bound of $O(\log^2 n)$ was also proven in [7] in a radio network model without asynchronous wake-up and in which nodes have a-priori knowledge about their neighborhood.

Finally, it is also possible to efficiently color the nodes of a network as shown in [17], and subsequently improved and generalized in Chap. 12 of [15].

Theorem 4 *In the unstructured radio network model, a correct coloring with at most $O(\Delta)$ colors can be computed in time $O(\Delta \log n)$ with high probability.*

Similar bounds for a model with collision detection mechanisms are proven in [3].

Applications

In wireless ad hoc and sensor networks, local network coordination structures find important applications. In particular, clusterings and colorings can help in facilitating the communication between adjacent nodes (MAC layer protocols) and between distant nodes (routing protocols), or to improve the energy efficiency of the network.

The following mentions two specific examples of applications: Based on the MIS algorithms of Theorem 3, a protocol is presented in [5], which efficiently constructs a *spanner*, i. e., a more sophisticated initial infrastructure that helps in structuring wireless multi-hop network. In [16], the same MIS algorithm is used as an ingredient for a protocol that minimizes the energy consumption of wireless sensor nodes during the *deployment phase*, a problem that has been first studied in [14].

Recommended Reading

1. Alon, N., Bar-Noy, A., Linial, N., Peleg, D.: A Lower Bound for Radio Broadcast. J. Comput. Syst. Sci. **43**, 290–298 (1991)
2. Bar-Yehuda, R., Goldreich, O., Itai, A.: On the time-complexity of broadcast in radio networks: an exponential gap between determinism randomization. In: Proc. 6th Symposium on Principles of Distributed Computing (PODC), pp. 98–108 (1987)
3. Busch, R., Magdon-Ismail, M., Sivrikaya, F., Yener, B.: Contention-Free MAC Protocols for Wireless Sensor Networks. In: Proc. 18th Annual Conference on Distributed Computing (DISC) (2004)
4. Chrobak, M., Gąsieniec, L., Kowalski, D.: The Wake-Up Problem in Multi-Hop Radio Networks. In: Proc. of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 992–1000 (2004)
5. Farach-Colton, M., Fernandes, R.J., Mosteiro, M.A.: Bootstrapping a Hop-Optimal Network in the Weak Sensor Model. In: Proc. of the 13th European Symposium on Algorithms (ESA), pp. 827–838 (2005)
6. Farach-Colton, M., Fernandes, R.J., Mosteiro, M.A.: Lower Bounds for Clear Transmissions in Radio Networks. In: Proc. of the 7th Latin American Symposium on Theoretical Informatics (LATIN), pp. 447–454 (2006)
7. Gandhi, R., Parthasarathy, S.: Distributed Algorithms for Coloring and Connected Domination in Wireless Ad Hoc Networks. In: Foundations of Software Technology and Theoretical Computer Science (FSTTCS), pp. 447–459 (2004)
8. Gąsieniec, L., Pelc, A., Peleg, D.: The Wakeup Problem in Synchronous Broadcast Systems (Extended Abstract). In: Proc. of the 19th ACM Symposium on Principles of Distributed Computing (PODC), pp. 113–121 (2000)
9. Jurdziński, T., Stachowiak, G.: Probabilistic Algorithms for the Wakeup Problem in Single-Hop Radio Networks. In: Proc. of the 13th Annual International Symposium on Algorithms and Computation (ISAAC), pp. 535–549 (2002)
10. Kuhn, F., Moscibroda, T., Wattenhofer, R.: Initializing Newly Deployed Ad Hoc and Sensor Networks. In: Proc. of the 10th Annual International Conference on Mobile Computing and Networking (MOBICOM), pp. 260–274 (2004)
11. Kuhn, F., Moscibroda, T., Wattenhofer, R.: What Cannot Be Computed Locally! In: Proceedings of 23rd Annual Symposium on Principles of Distributed Computing (PODC), pp. 300–309 (2004)
12. Linial, N.: Locality in Distributed Graph Algorithms. SIAM J. Comput. **21**(1), 193–201 (1992)
13. Luby, M.: A Simple Parallel Algorithm for the Maximal Independent Set Problem. SIAM J. Comput. **15**, 1036–1053 (1986)
14. McGlynn, M.J., Borbash, S.A.: Birthday Protocols for Low Energy Deployment and Flexible Neighborhood Discovery in Ad Hoc Wireless Networks. In: Proc. of the 2nd ACM Int. Symposium on Mobile Ad Hoc Networking & Computing (MOBIHOC), (2001)
15. Moscibroda, T.: Locality, Scheduling, and Selfishness: Algorithmic Foundations of Highly Decentralized Networks. Doctoral Thesis Nr. 16740, ETH Zurich (2006)
16. Moscibroda, T., von Rickenbach, P., Wattenhofer, R.: Analyzing the Energy-Latency Trade-off during the Deployment of Sensor Networks. In: Proc. of the 25th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), (2006)
17. Moscibroda, T., Wattenhofer, R.: Coloring Unstructured Radio Networks. In: Proc. of the 17th ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 39–48 (2005)
18. Moscibroda, T., Wattenhofer, R.: Maximal Independent Sets in Radio Networks. In: Proc. of the 23rd ACM Symposium on Principles of Distributed Computing (PODC), pp. 148–157 (2005)

Local Search Algorithms for *k*SAT

1999; Schöning

KAZUO IWAMA
School of Informatics, Kyoto University, Kyoto, Japan

Problem Definition

The CNF Satisfiability problem is to determine, given a CNF formula F with n variables, whether or not there exists a satisfying assignment for F. If each clause of F contains at most k literals, then F is called a k-CNF formula and the problem is called k-SAT, which is one of the most fundamental NP-complete problems. The trivial algorithm is to search 2^n 0/1-assignments for the n variables. But since [6], several algorithms which run significantly faster than this $O(2^n)$ bound have been developed. As a simple exercise, consider the following straightforward algorithm for 3-SAT, which gives us an upper bound of 1.913^n: Choose an arbitrary clause in F, say $(x_1 \vee \overline{x_2} \vee x_3)$. Then generate seven new formulas by substituting to these x_1, x_2 and x_3 all the possible values excepting $(x_1, x_2, x_3) = (0, 1, 0)$ which obviously unsatisfies F. Now one can check the satisfiability of these seven formulas and conclude that F is satisfiable iff at least one of them is satisfiable. (Let $T(n)$ denote the time complexity of this algorithm. Then one can get the recurrence $T(n) \leq 7 \times T(n-3)$ and the above bound follows.)

Key Results

In the long history of k-SAT algorithms, the one by Schöning [11] is an important breakthrough. It is a standard local search and the algorithm itself is not new (see e. g. [7]). Suppose that y is the current assignment (its initial value is selected uniformly at random). If y is a satisfying assignment, then the algorithm answers yes and terminates. Otherwise, there is at least one clause whose three literals are all false under y. Pick an arbitrary such clause and select one of the three literals in it at random. Then flip (true to false and vice versa) the value of that variable, replace y with that new assignment and then repeat the same procedure. More formally:

SCH(CNF-formula F, integer I)
  repeat I times
    y = uniformly random vector $\in \{0,1\}^n$
    z = **RandomWalk**(F, y);
    if z satisfies F
      then output(z); exit;
  end
  output('Unsatisfiable');
RandomWalk(CNF formula $G(x_1, x_2, \ldots, x_n)$, assignment y);
  $y' = y$;
  for $3n$ times
    if y' satisfies G
      then return y'; exit;
    $C \leftarrow$ an arbitrary clause of G that is not satisfied by y';
    Modify y' as follows:
      select one literal of C uniformly at random and flip the assignment to this literal;
  end
  return y'

Schöning's analysis of this algorithm is very elegant. Let $d(a, b)$ denote the Hamming distance between two binary vectors (assignments) a and b. For simplicity, suppose that the formula F has only one satisfying assignment y^* and the current assignment y is far from y^* by Hamming distance d. Suppose also that the currently false clause C includes three variables, x_i, x_j and x_k. Then y and y^* must differ in at least one of these three variables. This means that if the value of x_i, x_j or x_k is flipped, then the new assignment gets closer to y^* by Hamming distance one with probability at least 1/3. Also, the new assignment gets farther by Hamming distance one with probability at most 2/3. The argument can be generalized to the case that F has multiple satisfying assignments. Now here comes the key lemma:

Lemma 1 *Let F be a satisfiable formula and y^* be a satisfying assignment for F. For each assignment y, the probability that a satisfying assignment (that may be different from y^*) is found by* ***RandomWalk***(F, y) *is at least $(1/(k-1))^{d(y,y^*)}/p(n)$, where $p(n)$ is a polynomial in n.*

By taking the average over random initial assignments, the following theorem follows:

Theorem 2 *For any satisfiable formula F on n variables, the success probability of* ***RandomWalk***(F, y) *is at least $(k/2(k-1))^n/p(n)$ for some polynomial p. Thus, by setting $I = (2(k-1)/k)^n \cdot p(n)$,* ***SCH*** *finds a satisfying assignment with high probability. When $k = 3$, this value of I is $O(1.334^n)$.*

Applications

The Schöning's result has been improved by a series of papers [1,3,9] based on the idea of [3]. Namely, Random Walk is combined with the (polynomial time) 2SAT algorithm, which makes it possible to choose better initial assignments. For derandomization of **SCH**, see [2]. [4] developed a nontrivial combination of **SCH** with another famous, backtrack-type algorithm by [8], resulting in the then fastest algorithm with $O(1.324^n)$ running time. The current fastest algorithm is due to [10], which

is based on the same approach as [4] and runs in time $O(1.32216^n)$.

Open Problems

k-SAT is probably the most popular NP-complete problem for which numerous researchers are competing for its fastest algorithm. Thus improving its time bound is always a good research target.

Experimental Results

AI researchers have also been very active in SAT algorithms including local search, see e. g. [5].

Cross References

- ▶ Exact Algorithms for General CNF SAT
- ▶ Random Planted 3-SAT

Recommended Reading

1. Baumer, S., Schuler, R.: Improving a probabilistic 3-SAT algorithm by dynamic search and independent clause pairs. ECCC TR03-010, (2003) Also presented at SAT (2003)
2. Dantsin, E., Goerdt, A., Hirsch, E.A., Kannan, R., Kleinberg, J., Papadimitriou, C., Raghavan, P., Schöning, U.: A deterministic $(2 - 2/(k+1))^n$ algorithm for *k*-SAT based on local search. Theor. Comput. Sci. **289**(1), 69–83 (2002)
3. Hofmeister, T., Schöning, U., Schuler, R., Watanabe, O.: Probabilistic 3-SAT algorithm further improved. Proceedings 19th Symposium on Theoretical Aspects of Computer Science. LNCS **2285**, 193–202 (2002)
4. Iwama, K., Tamaki, S.: Improved upper bounds for 3-SA T. In: Proceedings 15th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 321–322. New Orleans, USA (2004)
5. Kautz, H., Selman, B.: Ten Challenges Redux: Recent Progress in Propositional Reasoning and Search. Proceedings 9th International Conference on Principles and Practice of Constraint Programming, pp. 1–18. Kinsale, Ireland (2003)
6. Monien, B., Speckenmeyer, E.: Solving satisfiability in less than 2^n steps. Discret. Appl. Math. **10**, 287–295 (1985)
7. Papadimitriou, C.H.: On selecting a satisfying truth assignment. Proceedings 32nd Annual Symposium on Foundations of Computer Science, pp. 163–169. San Juan, Puerto Rico (1991)
8. Paturi, R., Pudlák, P., Saks, M.E., Zane, F.: An improved exponential-time algorithm for *k*-SAT. Proceedings 39th Annual Symposium on Foundations of Computer Science, pp. 628–637. Palo Alto, USA (1998) Also, J. ACM **52**(3), 337–364 (2006)
9. Rolf, D.: 3-SAT ∈ *RTIME*($O(1.32793^n)$). ECCC TR03–054. (2003)
10. Rolf, D.: Improved Bound for the PPSZ/Schöning-Algorithm for 3-SAT. J. Satisf. Boolean Model. Comput. **1**, 111–122 (2006)
11. Schöning, U.: A probabilistic algorithm for *k*-SAT and constraint satisfaction problems. Proceedings 40th Annual Symposium on Foundations of Computer Science, pp. 410–414. New York, USA (1999)

Local Search for *K*-medians and Facility Location

2001; Arya, Garg, Khandekar, Meyerson, Munagala, Pandit

KAMESH MUNAGALA
Levine Science Research Center, Duke University, Durham, NC, USA

Keywords and Synonyms

k-Medians; *k*-Means; *k*-Medioids; Facility location; Point location; Warehouse location; Clustering

Problem Definition

Clustering is a form of *unsupervised learning*, where the goal is to "learn" useful patterns in a data set $\mathcal{D}$ of size n. It can also be thought of as a data compression scheme where a large data set is represented using a smaller collection of "representatives". Such a scheme is characterized by specifying the following:

1. A *distance* metric $\mathbf{d}$ between items in the data set. This metric should satisfy the triangle inequality: $\mathbf{d}(i,j) \leq \mathbf{d}(j,k) + \mathbf{d}(k,i)$ for any three items $i, j, k \in \mathcal{D}$. In addition, $\mathbf{d}(i,j) = \mathbf{d}(j,i)$ for all $i, j \in S$ and $\mathbf{d}(i,i) = 0$. Intuitively, if the distance between two items is smaller, they are more similar. The items are usually points in some high dimensional Euclidean space $\mathcal{R}^d$. The commonly used distance metrics include the Euclidean and Hamming metrics, and the cosine metric measuring the angle between the vectors representing the items.
2. The output of the clustering process is a partitioning of the data. This chapter deals with *center-based* clustering. Here, the output is a smaller set $C \subset \mathcal{R}^d$ of *centers* which best represents the input data set $S \subset \mathcal{R}^d$. It is typically the case that $|C| \ll |\mathcal{D}|$. Each item $j \in \mathcal{D}$ is *mapped to* or *approximated by* the the closest center $i \in C$, implying $\mathbf{d}(i,j) \leq \mathbf{d}(i',j)$ for all $i' \in C$. Let $\sigma : \mathcal{D} \to C$ denote this mapping. This is intuitive since closer-by (similar) items will be mapped to the same center.
3. A measure of the *quality* of the clustering, which depends on the desired output. There are several commonly used measures for the quality of clustering. In each of the clustering measures described below, the goal is to choose C such that $|C| = k$ and the objective function $f(C)$ is minimized.

***k*-center:** $f(C) = \max_{j \in \mathcal{D}} \mathbf{d}(j, \sigma(j))$.
***k*-median:** $f(C) = \sum_{j \in \mathcal{D}} \mathbf{d}(j, \sigma(j))$.
***k*-means:** $f(C) = \sum_{j \in \mathcal{D}} \mathbf{d}(j, \sigma(j))^2$.

All the objectives described above are NP-HARD to optimize in general metric spaces **d**, leading to the study of heuristic and approximation algorithms. In the rest of this chapter, the focus is on the k-median objective. The approximation algorithms for k-median clustering are designed for **d** being a general possibly non-Euclidean metric space. In addition, a collection $\mathcal{F}$ of possible center locations is given as input, and the set of centers C is restricted to $C \subseteq \mathcal{F}$. From the perspective of approximation, the restriction of the centers to a finite set $\mathcal{F}$ is not too restrictive – for instance, the optimal solution which is restricted to $\mathcal{F} = \mathcal{D}$ has objective value at most a factor 2 of the optimal solution which is allowed arbitrary $\mathcal{F}$. Denote $|\mathcal{D}| = n$, and $|\mathcal{F}| = m$. The running times of the heuristics designed will be polynomial in $m\,n$, and a parameter $\varepsilon > 0$. The metric space **d** is now defined over $\mathcal{D} \cup \mathcal{F}$.

A related problem to k-medians is its Lagrangean relaxation, called FACILITY LOCATION. In this problem, there is a again collection $\mathcal{F}$ of possible center locations. Each location $i \in \mathcal{F}$ has a location cost r_i. The goal is to choose a collection $C \subseteq \mathcal{F}$ of centers and construct the mapping $\sigma : S \to C$ from the items to the centers such that the following function is minimized:

$$f(C) = \sum_{j \in \mathcal{D}} \mathbf{d}(j, \sigma(j)) + \sum_{i \in C} r_i \ .$$

The facility location problem effectively gets rid of the hard bound k on the number of centers in k-medians, and replaces it with the center cost term $\sum_{i \in C} r_i$ in the objective function, thereby making it a Lagrangean relaxation of the k-median problem. Note that the costs of centers can now be non-uniform.

The approximation results for both the k-median and facility location problems carry over as is to the weighted case: Each item $j \in \mathcal{D}$ is allowed to have a non-negative weight w_j. In the objective function $f(C)$, the term $\sum_{j \in \mathcal{D}} \mathbf{d}(j, \sigma(j))$ is replaced with $\sum_{j \in \mathcal{D}} w_j \cdot \mathbf{d}(j, \sigma(j))$. The weighted case is especially relevant to the FACILITY LOCATION problem where the item weights signify user demands for a resource, and the centers denote locations of the resource. In the remaining discussion, "items" and "users" are used inter-changably to denote members of the set $\mathcal{D}$.

Key Results

The method of choice for solving both the k-median and FACILITY LOCATION problems are the class of local search heuristics, which run in "local improvement" steps. At each step t, the heuristic maintains a set C_t of centers. For the k-median problem, this collection satisfies $|C_t| = k$. A local improvement step first generates a collection of new solutions $\mathcal{E}_{t+1}$ from C_t. This is done such that $|\mathcal{E}_{t+1}|$ is polynomial in the input size. For the k-median problem, in addition, each $C \in \mathcal{E}_{t+1}$ satisfies $|C| = k$. The improvement step sets $C_{t+1} = \text{argmin}_{C \in \mathcal{E}_{t+1}} f(C)$. For a prespecified parameter $\varepsilon > 0$, the improvement iterations stop at the first step T where $f(C_T) \geq (1 - \varepsilon) f(C_{T-1})$.

The key design issue is the specification of the start set C_0, and the construction of $\mathcal{E}_{t+1}$ from C_t. The key analysis issues are bounding the number of steps T till termination, and the quality of the final solution $f(C_T)$ against the optimal solution $f(C^*)$. The ratio $(f(C_T))/(f(C^*))$ is termed the "locality gap" of the heuristic.

Since each improvement step reduces the value of the solution by at least a factor of $(1 - \varepsilon)$, the running time in terms of number of improvement steps is given by the following expression (here D is the ratio of the largest to smallest distance in the metric space over $\mathcal{D} \cup \mathcal{F}$).

$$T \leq \log_{1/(1-\varepsilon)} \left(\frac{f(C_0)}{f(C_T)} \right) \leq \frac{\log \left(\frac{f(C_0)}{f(C_T)} \right)}{\varepsilon} \leq \frac{\log(nD)}{\varepsilon}$$

which is polynomial in the input size. Each improvement step needs computation of $f(C)$ for $C \in \mathcal{E}_t$. This is polynomial in the input size since $|\mathcal{E}_t|$ is assumed to be polynomial.

k-Medians

The first local search heuristic with provable performance guarantees is presented in the work of Arya et al. [1]. The is the natural p-swap heuristic: Given the current center set C_t of size k, the set $\mathcal{E}_{t+1}$ is defined by:

$$\mathcal{E}_{t+1} = \{(C_t \setminus \mathcal{A}) \cup \mathcal{B} \ , \quad \text{where } \mathcal{A} \subseteq C_t, \mathcal{B} \subseteq \mathcal{F} \setminus C_t, |\mathcal{A}| = |\mathcal{B}| \leq p\} \ .$$

The above simply means swap at most p centers from C_t with the same number of centers from $\mathcal{F} \setminus C_t$. Recall that $|\mathcal{D}| = n$ and $|\mathcal{F}| = m$. Clearly, $|\mathcal{E}_{t+1}| \leq (k(m-k))^p \leq (km)^p$. The start set C_0 is chosen arbitrarily. The value p is a parameter which affects the running time and the approximation ratio. It is chosen to be a constant, so that $|\mathcal{E}_t|$ is polynomial in m.

Theorem 1 ([1]) *The p-swap heuristic achieves locality gap $(3 + 2/p) + \varepsilon$ in running time $O(nk(\log(nD))/\varepsilon(mk)^p)$. Furthermore, for every p there is a k-median instance where the p-swap heuristic has locality gap exactly $(3 + 2/p)$.*

Setting $p = 1/\varepsilon$, the above heuristic achieves a $3 + \varepsilon$ approximation in running time $\tilde{O}(n(mk)^{O(1/\varepsilon)})$.

Facility Location

For this problem, since there is no longer a constraint on the number of centers, the local improvement step needs to be suitably modified. There are two local search heuristics both of which yield a locality gap of $3 + \varepsilon$ in polynomial time.

The "add/delete/swap" heuristic proposed by Kuehn and Hamburger [10] either adds a center to C_t, drops a center from C_t, or swaps a center in C_t with one in $\mathcal{F} \setminus C_t$. The start set C_0 is again arbitrary.

$$\mathcal{E}_{t+1} = \{(C_t \setminus \mathcal{A}) \cup \mathcal{B}, \text{ where } \mathcal{A} \subseteq C_t, \mathcal{B} \subseteq \mathcal{F} \setminus C_t, \\ |\mathcal{A}| = 0, |\mathcal{B}| = 1 \text{ or } |\mathcal{A}| = 1, |\mathcal{B}| = 0, \text{ or } |\mathcal{A}| = 1, |\mathcal{B}| = 1\}$$

Clearly, $|\mathcal{E}_{t+1}| = O(m^2)$, making the running time polynomial in the input size and $1/\varepsilon$. Korupolu, Plaxton, and Rajaraman [9] show that this heuristic achieves a locality gap of at most $5 + \varepsilon$. Arya et al. [1] strengthen this analysis to show that this heuristic achieves a locality gap of $3 + \varepsilon$, and that bound this is tight in the sense that there are instances where the locality gap is exactly 3.

The "add one/delete many" heuristic proposed by Charikar and Guha [2] is slightly more involved. This heuristic adds one facility and drops all facilities which become irrelevant in the new solution.

$$\mathcal{E}_{t+1} = \{(C_t \cup \{i\}) \setminus I(i), \text{ where } i \in \mathcal{F} \setminus C_t, I(i) \subseteq C_t\}$$

The set $I(i)$ is computed as follows: Let W denote the set of items closer to i than to their assigned centers in C_t. These items are ignored from the computation of $I(i)$. For every center $s \in C_t$, let U_s denote all items which are assigned to s. If $f_s + \sum_{j \in U_s \setminus W} d_j \mathbf{d}(j, s) > \sum_{j \in U_s \setminus W} d_j \mathbf{d}(j, i)$, then it is cheaper to remove location s and reassign the items in $U_s \setminus W$ to i. In this case, s is placed in $I(i)$. Let N denote $m + n$. Computing $I(i)$ is therefore a $O(N)$ time greedy procedure, making the overall running time polynomial. Charikar and Guha [2] show the following theorem:

Theorem 2 ([2]) *The local search heuristic which attempts to add a random center $i \notin C_t$ and remove set $I(i)$, computes a $3 + \varepsilon$ approximation with high probability within $T = O(N \log N(\log N + 1/\varepsilon))$ improvement steps, each with running time $O(N)$.*

Capacitated Variants

Local search heuristics are also known for capacitated variants of the *k*-median and facility location problems. In this variant, each possible location $i \in \mathcal{F}$ can serve at most u_i number of users. In the soft capacitated variant of facility location, some $r_i \geq 0$ copies can be opened at $i \in \mathcal{F}$ so that the facility cost is $f_i r_i$ and the number of users served is at most $r_i u_i$. The optimization goal is now to decide the value of r_i for each $i \in \mathcal{F}$ so that the assignment of users to the centers satisfies the capacity constraints at each center, and the cost of opening the centers and assigning the users is minimized. For this variant, Arya *et al.* [1] show a local search heuristic with a locality gap of $4 + \varepsilon$.

In the version of facility location with hard capacities, location $i \in \mathcal{F}$ has a hard bound u_i on the number of users that can be assigned here. If all the capacities u_i are equal (uniform case), Korupolu, Plaxton, and Rajaraman [9] present an elegant local search heuristic based on solving a transshipment problem which achieves a $8 + \varepsilon$ locality gap. The analysis is improved by Chudak and Williamson [4] to show a locality gap $6 + \varepsilon$. The case of non-uniform capacities requires significantly new ideas – Pál, Tardos, and Wexler [14] present a network flow based local search heuristic that achieves a locality gap of $9 + \varepsilon$. This bound is improved to $8 + \varepsilon$ by Mahdian and Pál [12], who generalize several of the local search techniques described above in order to obtain a constant factor approximation for the variant of facility location where the facility costs are arbitrary non-decreasing functions of the demands they serve.

Related Algorithmic Techniques

Both the *k*-median and facility location problems have a rich history of approximation results. Since the study of uncapacitated facility location was initiated by Cornuejols, Nemhauser, and Wolsey [5], who presented a natural linear programming (LP) relaxation for this problem, several constant-factor approximations have been designed via several techniques, ranging from rounding of the LP solution [11,15], local search [2,9], the primal-dual schema [7], and dual fitting [6]. For the *k*-median problem, the first constant factor approximation [3] of $6\frac{2}{3}$ was obtained by rounding the natural LP relaxation via a generalization of the filtering technique in [11]. This result was subsequently improved to a 4 approximation by Lagrangean relaxation and the primal-dual schema [2,7], and finally to a $(3 + \varepsilon)$ approximation via local search [1].

Applications

The facility location problem has been widely studied in operations research [5,10], and forms a fundamental primitive for several resource location problems. The *k*-medians and *k*-means metrics are widely used in clustering, or unsupervised learning. For clustering applications, several heuristic improvements to the basic local search framework have been proposed: *k*-Medioids [8] selects

a random input point and replaces it with one of the existing centers if there is an improvement; the CLARA [8] implementation of k-Medioids chooses the centers from a random sample of the input points to speed up the computation; the CLARANS [13] heuristic draws a fresh random sample of feasible centers before each improvement step to further improve the efficiency.

Cross References

► Facility Location

Recommended Reading

1. Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., Pandit, V.: Local search heuristics for k-median and facility location problems. SIAM J. Comput. **33**(3), 544–562 (2004)
2. Charikar, M., Guha, S.: Improved combinatorial algorithms for facility location problems. SIAM J. Comput. **34**(4), 803–824 (2005)
3. Charikar, M., Guha, S., Tardos, É., Shmoys, D.B.: A constant-factor approximation algorithm for the k-median problem (extended abstract). In: STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing, pp. 1–10. Atlanta, May 1-4 1999
4. Chudak, F.A., Williamson, D.P.: Improved approximation algorithms for capacitated facility location problems. Math. Program. **102**(2), 207–222 (2005)
5. Cornuejols, G., Nemhauser, G.L., Wolsey, L.A.: The uncapacitated facility location problem. In: Discrete Location Theory, pp. 119–171. Wiley, New York (1990)
6. Jain, K., Mahdian, M., Markakis, E., Saberi, A., Vazirani, V.V.: Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. J. ACM **50**(6), 795–824 (2003)
7. Jain, K., Vazirani, V.V.: Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. J. ACM **48**(2), 274–296 (2001)
8. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. Wiley, New York (1990)
9. Korupolu, M.R., Plaxton, C.G., Rajaraman, R.: Analysis of a local search heuristic for facility location problems. In: SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms, pp. 1–10. San Francisco, USA; 25–26 January 1998
10. Kuehn, A.A., Hamburger, M.J.: A heuristic program for locating warehouses. Management Sci. **9**(4), 643–666 (1963)
11. Lin, J.-H., Vitter, J.S.: ε-approximations with minimum packing constraint violation (extended abstract). In: STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing, pp. 771–782. Victoria (1992)
12. Mahdian, M., Pál, M.: Universal facility location. In: European Symposium on Algorithms, pp. 409–421. Budapest, Hungary, September 16–19 2003
13. Ng, R.T., Han, J.: Efficient and effective clustering methods for spatial data mining. In: Proc. Symp. on Very Large Data Bases (VLDB), pp. 144–155. Santiago de Chile, 12–15 September 1994
14. Pál, M., Tardos, É., Wexler, T.: Facility location with nonuniform hard capacities. In: Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, pp. 329–338. Las Vegas, 14–17 October 2001
15. Shmoys, D.B., Tardos, É., and Aardal, K.: Approximation algorithms for facility location problems. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, pp. 265–274. El Paso, 4–6 May 1997

Location-Based Routing

► Geographic Routing
► Routing in Geometric Networks

Lower Bounds for Dynamic Connectivity

2004; Pătraşcu, Demaine

MIHAI PĂTRAŞCU
CSAIL, MIT, Cambridge, MA, USA

Keywords and Synonyms

Dynamic trees

Problem Definition

The dynamic connectivity problem requests maintenance of a graph G subject to the following operations:

`insert`(u, v)**:** insert an undirected edge (u, v) into the graph.
`delete`(u, v)**:** delete the edge (u, v) from the graph.
`connected`(u, v)**:** test whether u and v lie in the same connected component.

Let m be an upper bound on the number of edges in the graph. This entry discusses cell-probe lower bounds for this problem. Let t_u be the complexity of `insert` and `delete` and t_q the complexity of `query`.

The Partial-Sums Problem

Lower bounds for dynamic connectivity are intimately related to lower bounds for another classic problem: maintaining partial sums. Formally, the problem asks one to maintain an array $A[1..n]$ subject to the following operations:

`update`(k, Δ)**:** let $A[k] \leftarrow \Delta$.
`sum`(k)**:** returns the partial sum $\sum_{i=1}^{k} A[i]$.
`testsum`(k, σ)**:** returns a boolean value indicating whether $\texttt{sum}(k) = \sigma$.

To specify the problem completely, let elements $A[i]$ come from an arbitrary group G containing at least 2^δ elements. In the cell-probe model with b-bit cells, let t_u^Σ be the complexity of `update` and t_q^Σ the complexity of `testsum` (which is also a lower bound on `sum`).

The tradeoffs between t_u^Σ and t_q^Σ are well understood for all values of b and δ. However, this entry only considers lower bounds under the standard assumptions that $b = \Omega(\lg n)$ and $t_u \geq t_q$. It is standard to assume $b = \Omega(\lg n)$ for upper bounds in the RAM model; this assumption also means that the lower bound applies to the pointer machine. Then, Pătraşcu and Demaine [6] prove:

Theorem 1 *The complexity of the partial-sums problems satisfies:* $t_q^\Sigma \cdot \lg(t_u^\Sigma / t_q^\Sigma) = \Omega(\delta/b \cdot \lg n)$.

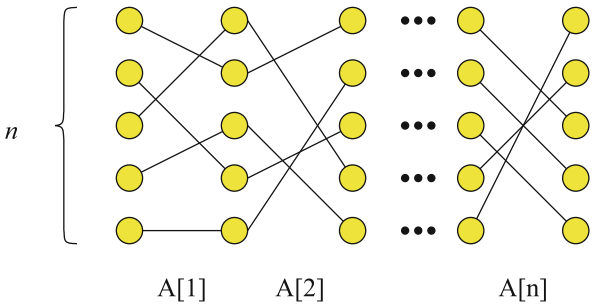
Observe that this matches the textbook upper bound using augmented trees. One can build a balanced binary tree over $A[1], \ldots, A[n]$ and store in every internal node the sum of its subtree. Then, updates and queries touch $O(\lg n)$ nodes (and spend $O(\lceil \delta/b \rceil)$ time in each one due to the size of the group). To decrease the query time, one can use a B-tree.

Relation to Dynamic Connectivity

We now clarify how lower bounds for maintaining partial sums imply lower bounds for dynamic connectivity. Consider the partial-sums problem over the group $G = S_n$, i. e., the permutation group on n elements. Note that $\delta = \lg(n!) = \Omega(n \lg n)$. It is standard to set $b = \Theta(\lg n)$, as this is the natural word size used by dynamic connectivity upper bounds. This implies $t_q^\Sigma \lg(t_u^\Sigma / t_q^\Sigma) = \Omega(n \lg n)$.

The lower bound follows from implementing the partial-sums operations using dynamic connectivity operations. Refer to Fig. 1. The vertices of the graph form an integer grid of size $n \times n$. Each vertex is incident to at most two edges, one edge connecting to a vertex in the previous column and one edge connecting to a vertex in the next column. Point (x, y_1) in the grid is connected to point $(x+1, A[x](y_1))$, i. e.,the edges between two adjacent columns describe the corresponding permutation from the partial-sums vector.

To implement `update`(x, π), all the edges between column x and $x+1$ are first deleted and then new edges are inserted according to π. This gives $t_u^\Sigma = O(2n \cdot t_u)$. To implement `testsum`$(x, \pi)$, one can use n `connected` queries between the pairs of points $(1, y) \rightsquigarrow (x+1, \pi(y))$. Then, $t_q^\Sigma = O(n \cdot t_q)$. Observe that the `sum` query cannot be implemented as easily. Dynamic connectivity is the main motivation to study the `testsum` query.



Lower Bounds for Dynamic Connectivity, Figure 1
Constructing an instance of dynamic connectivity that mimics the partial-sums problem

The lower bound of Theorem 1 translates into $nt_q \cdot \lg(2nt_u/nt_q) = \Omega(n \lg n)$; hence $t_q \lg(t_u/t_q) = \Omega(\lg n)$. Note that this lower bound implies $\max\{t_u, t_q\} = \Omega(\lg n)$. The best known upper bound (using amortization and randomization) is $O(\lg n(\lg \lg n)^3)$ [9]. For any $t_u = \Omega(\lg n(\lg \lg n)^3)$, the lower bound tradeoff is known to be tight. Note that the graph in the lower bound is always a disjoint union of paths. This implies optimal lower bounds for two important special cases: dynamic trees [8] and dynamic connectivity in plane graphs [2].

Key Results

Understanding Hierarchies

Epochs To describe the techniques involved in the lower bounds, first consider the `sum` query and assume $\delta = b$. In 1989, Fredman and Saks [3] initiated the study of dynamic cell-probe lower bounds, essentially showing a lower bound of $t_q^\Sigma \lg t_u^\Sigma = \Omega(\lg n)$. Note that this implies $\max\{t_q^\Sigma, t_u^\Sigma\} = \Omega(\lg n / \lg \lg n)$.

At an intuitive level, their argument proceeded as follows. The hard instance will have n random updates, followed by one random query. Leave $r \geq 2$ to be determined. Looking *back* in time from the query, one groups the updates into exponentially growing epochs: the latest r updates are epoch 1, the earlier r^2 updates are epoch 2, etc. Note that epoch numbers increase going back in time, and there are $O(\log_r n)$ epochs in total.

For some epoch i, consider revealing to the query all updates performed in all epochs different from i. Then, the query reduces to a partial-sums query among the updates in epoch i. Unless the query is to an index below the minimum index updated in epoch i, the answer to the query is still uniformly random, i. e., has δ bits of entropy. Furthermore, even if one is given, say, $r^i \delta/100$ bits of information about epoch i, the answer still has $\Omega(\delta)$ bits of entropy on

average. This is because the query and updates in epoch i are uniformly random, so the query can ask for any partial sum of these updates, uniformly at random. Each of the r^i partial sums is an independent random variable of entropy δ.

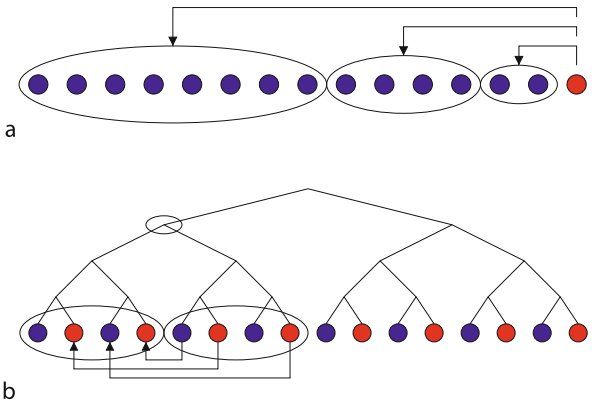
Now one can ask how much information is available to the query. At the time of the query, let each cell be associated with the epoch during which it was last written. Choosing an epoch i uniformly at random, one can make the following intuitive argument:

1. No cells written by epochs $i+1, i+2, \ldots$ can contain information about epoch i, as they were written in the past.
2. In epochs $1, \ldots, i-1$, a number of $bt_u^\Sigma \cdot \sum_{j=1}^{i-1} r^j \leq bt_u^\Sigma \cdot 2r^{i-1}$ bits were written. This is less than $r^i\delta/100$ bits of information for $r > 200t_u^\Sigma$ (recall the assumption $\delta = b$). By the above, this implies the query answer still has $\Omega(\delta)$ bits of entropy.
3. Since i is uniformly random among $\Theta(\log_r n)$ epochs, the query makes an expected $O(t_q^\Sigma/\log_r n)$ probes to cells from epoch i. All queries that make no cell probes to epoch i have a fixed answer (entropy 0), and all other queries have answers of entropy $\leq \delta$. Since an average query has entropy $\Omega(\delta)$, a query must probe a cell from epoch i with constant probability. That means $t_q^\Sigma/\log_r n = \Omega(1)$, and $\sum = \Omega(\log_r n) = \Omega(\lg n/\lg t_u^\Sigma)$.

One should appreciate the duality between the proof technique and the natural upper bounds based on a hierarchy. Consider an upper bound based on a tree of degree r. The last r random updates (epoch 1) are likely to be uniformly spread in the array. This means the updates touch different children of the root. Similarly, the r^2 updates in epoch 2 are likely to touch every node on level 2 of the tree, and so on. Now, the lower bound argues that the query needs to traverse a root-to-leaf path, probing a node on every level of the tree (this is equivalent to one cell from every epoch).

Time Hierarchies Despite considerable refinement to the lower bound techniques, the lower bound of $\Omega(\lg n/\lg\lg n)$ was not improved until 2004. Then, Pătraşcu and Demaine [6] showed an optimal bound of $t_q^\Sigma \lg(t_u^\Sigma/t_q^\Sigma) = \Omega(\lg n)$, implying $\max\{t_u^\Sigma, t_q^\Sigma\} = \Omega(\lg n)$. For simplicity, the discussion below disregards the tradeoff and just sketches the $\Omega(\lg n)$ lower bound.

Pătraşcu and Demaine's [6] counting technique is rather different from the epoch technique; refer to Fig. 2. The hard instance is a sequence of n operations alternating between updates and queries. They consider a balanced binary tree over the time axis, with every leaf being an operation. Now for every node of the tree, they propose to count the number of cell probes made in the right subtree to a cell written in the left subtree. Every probe is counted exactly once, for the lowest common ancestor of the read and write times.



Lower Bounds for Dynamic Connectivity, Figure 2
Analysis of cell probes in the a epoch-based and b time-hierarchy techniques

Now focus on two sibling subtrees, each containing k operations. The $k/2$ updates in the left subtree, and the $k/2$ queries in the right subtree, are expected to interleave in index space. Thus, the queries in the right subtree ask for $\Omega(k)$ different partial sums of the updates in the left subtree. Thus, the right subtree "needs" $\Omega(k\delta)$ bits of information about the left subtree, and this information can only come from cells written in the left subtree and read in the right one. This implies a lower bound of $\Omega(k)$ probes, associated with the parent of the sibling subtrees. This bound is linear in the number of leaves, so summing up over the tree, one obtains a total $\Omega(n \lg n)$ lower bound, or $\Omega(\lg n)$ cost per operation.

An Optimal Epoch Construction Rather surprisingly, Pătraşcu and Tarniţă [7] managed to reprove the optimal tradeoff of Theorem 1 with minimal modifications to the epoch argument. In the old epoch argument, the information revealed by epochs $1, \ldots, i-1$ about epoch i was bounded by the number of cells written in these epochs. The key idea is that an equally good bound is the number of cells read during epochs $1, \ldots, i-1$ and written during epoch i.

In principle, all cell reads from epoch $i-1$ could read data from epoch i, making these two bounds identical. However, one can randomize the epoch construction by inserting the query after an unpredictable number of updates. This randomization "smooths" out the distribution of epochs from which cells are read, i. e., a query

reads $O(t_q^\Sigma / \log_r n)$ cells from every epoch, in expectation over the randomness in the epoch construction. Then, the $O(r^{i-1})$ updates in epochs $1, \ldots, i-1$ only read $O(r^{i-1} \cdot t_u^\Sigma / \log_r n)$ cells from epoch i. This is not enough information if $r \gg t_u^\Sigma / \log_r n = \Theta(t_u^\Sigma / t_q^\Sigma)$, which implies $t_q^\Sigma = \Omega(\log_r n) = \Omega(\lg n / \lg(t_u^\Sigma / t_q^\Sigma))$.

Technical Difficulties

Nondeterminism The lower bounds sketched above are based on the fact that the `sum` query needs to output $\Omega(\delta)$ bits of information about every query. If dealing with the *decision* `testsum` query, an argument based on output entropy can no longer work.

The most successful idea for decision queries has been to convert them to queries with nonboolean output, in an extended cell-probe model that allows nondeterminism. In this model, the query algorithm is allowed to spawn an arbitrary number of computation threads. Each thread can make t_q cell probes, after with it must either terminate with a 'reject' answer, or return an answer to the query. All nonrejecting threads must return the same output. In this model, a query with arbitrary output is equivalent to a decision query, because one can just nondeterministically guess the answer, and then verify it.

By the above, the challenge is to prove good lower bounds for `sum` even in the nondeterminstic model. Nondeterminism shakes our view that when analyzing epoch i, only cell probes to epoch i matter. The trouble is that the query may not know *which* of its probes are actually to epoch i. A probe that reads a cell from a previous epoch provides at least some information about epoch i: no update in the epoch decided to overwrite the cell. Earlier this was not a problem because the goal was only to rule out the case that there are *zero* probes to epoch i. Now, however, different threads can probe any cell in memory, and one cannot determine which threads actually avoid probing anything in epoch i. In other words, there is a covert communication channel between epoch i and the query in which the epoch can use the choice of which cells to write in order to communicate information to the query.

There are two main strategies for handling nondeterministic query algorithms. Husfeldt and Rauhe [4] give a proof based on some interesting observations about the combinatorics of nondeterministic queries. Pătraşcu and Demaine [6] use the power of nondeterminism itself to output a small certificate that rules out useless cell probes. The latter result implies the optimal lower bound of Theorem 1 for `testsum` and, thus, the logarithmic lower bound for dynamic connectivity.

Alternative Histories The framework described above relies on fixing all updates in epochs different from i to an average value and arguing that the query answer still has a lot of variability, depending on updates in epoch i. This is true for aggregation problems but not for search problems. If a searched item is found with equal probability in any epoch, then fixing all other epochs renders epoch i irrelevant with probability $1 - 1/(\log_r n)$.

Alstrup et al. [1] propose a very interesting refinement to the technique, proving $\Omega(\lg n / \lg \lg n)$ lower bounds for an impressive collection of search problems. Intuitively, their idea is to consider $O(\log_r n)$ alternative histories of updates, chosen independently at random. Epoch i is relevant in at least one of the histories with constant probability. On the other hand, even if one knows what epochs 1 through $i-1$ learned about epoch i in *all histories*, answering a random query is still hard.

Bit-Probe Complexity Intuitively, if the word size is $b = 1$, the lower bound for connectivity should be roughly $\Omega(\lg^2 n)$, because a query needs $\Omega(\lg n)$ bits from every epoch. However, ruling out anything except zero probes to an epoch turns out to be difficult, for the same reason that the nondeterministic case is difficult. Without giving a very satisfactory understanding of this issue, Pătraşcu and Tarniţă [7] use a large bag of tricks to show an $\Omega((\lg n / \lg \lg n)^2)$ lower bound for dynamic connectivity. Furthermore, they consider the partial-sums problem in $\mathbb{Z}_2$ and show an $\Omega(\lg n / \lg \lg \lg n)$ lower bound, which is a triply-logarithmic factor away from the upper bound!

Applications

The lower bound discussed here extends by easy reductions to virtually all natural fully dynamic graph problems [6].

Open Problems

By far, the most important challenge for future research is to obtain a lower bound of $\omega(\lg n)$ per operation for some dynamic data structure in the cell-probe model with word size $\Theta(\lg n)$. Miltersen [5] specifies a set of technical conditions for what qualifies as a solution to such a challenge. In particular, the problem should be a *dynamic language membership* problem.

For the partial-sums problem, though `sum` is perfectly understood, `testsum` still lacks tight bounds for certain ranges of parameters [6]. In addition, obtaining tight bounds in the bit-probe model for partial sums in $\mathbb{Z}_2$ appears to be rather challenging.

Recommended Reading

1. Alstrup, S., Husfeldt, T., Rauhe, T.: Marked ancestor problems. In: Proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS), 1998, pp. 534–543
2. Eppstein, D., Italiano, G.F., Tamassia, R., Tarjan, R.E., Westbrook, J.R., Yung, M.: Maintenance of a minimum spanning forest in a dynamic planar graph. J. Algorithms **13**, 33–54 (1992). See also SODA'90
3. Fredman, M.L., Saks, M.E.: The cell probe complexity of dynamic data structures. In: Proc. 21st ACM Symposium on Theory of Computing (STOC), 1989, pp. 345–354
4. Husfeldt, T., Rauhe, T.: New lower bound techniques for dynamic partial sums and related problems. SIAM J. Comput. **32**, 736–753 (2003). See also ICALP'98
5. Miltersen, P.B.: Cell probe complexity - a survey. In: 19th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 1999 (Advances in Data Structures Workshop)
6. Pătraşcu, M. and Demaine, E.D.: Logarithmic lower bounds in the cell-probe model. SIAM J. Comput. **35**, 932–963 (2006). See also SODA'04 and STOC'04
7. Pătraşcu, M., Tarniţă, C.: On dynamic bit-probe complexity. Theor. Comput. Sci. **380**, 127–142 (2007). See also ICALP'05
8. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. J. Comput. Syst. Sci. **26**, 362–391 (1983). See also STOC'81
9. Thorup, M.: Near-optimal fully-dynamic graph connectivity. In: Proc. 32nd ACM Symposium on Theory of Computing (STOC), 2000, pp. 343–350

Low Stretch Spanning Trees

2005; Elkin, Emek, Spielman, Teng

Michael Elkin
Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel

Keywords and Synonyms

Spanning trees with low average stretch

Problem Definition

Consider a weighted connected multigraph $G = (V, E, \omega)$, where ω is a function from the edge set E of G into the set of positive reals. For a path P in G, the *weight* of P is the sum of weights of edges that belong to the path P. For a pair of vertices $u, v \in V$, the *distance* between them in G is the minimum weight of a path connecting u and v in G. For a spanning tree T of G, the stretch of an edge $(u, v) \in E$ is defined by

$$stretch_T(u, v) = \frac{dist_T(u, v)}{dist_G(u, v)} ,$$

and the average stretch over all edges of E is

$$avestr(G, T) = \frac{1}{|E|} \sum_{(u,v)\in E} stretch_T(u, v) .$$

The average stretch of a multigraph $G = (V, E, \omega)$ is defined as the smallest average stretch of a spanning tree T of G, $avestr(G, T)$. The average stretch of a positive integer n, $avestr(n)$, is the maximum average stretch of an n-vertex multigraph G. The problem is to analyze the asymptotic behavior of the function $avestr(n)$.

A closely related (dual) problem is to construct a probability distribution $\mathcal{D}$ of spanning trees for G, so that

$$expstr(G, \mathcal{D}) = \max_{e=(u,v)\in E} \mathbb{E}_{T\in\mathcal{D}}(stretch_T(u, v))$$

is small as possible. Analogously, $expstr(G) = \min_{\mathcal{D}}\{expstr(G, \mathcal{D})\}$, where the minimum is over all distributions $\mathcal{D}$ of spanning trees of G, and $expstr(n) = \max_G\{expstr(G)\}$, where the maximum is over all n-vertex multigraphs.

By viewing the problem as a 2-player zero-sum game between a tree player that aims to minimize the payoff, and an edge player that aims to maximize it, it is easy to see that for every positive integer n, $avestr(n) = expstr(n)$ [2]. The probabilistic version of the problem is, however, particularly convenient for many applications.

Key Results

The problem was studied since sixties [8,13,15,16]. A major progress in its study was achieved by Alon et al. [2], who showed that

$$\Omega(\log n) = avestr(n) = expstr(n)$$
$$= exp(O(\sqrt{\log n \cdot \log\log n})) .$$

Elkin et al. [9] improved the upper bound and showed that

$$avestr(n) = expstr(n) = O(\log^2 n \cdot \log\log n) .$$

Applications

One application of low stretch spanning trees is for solving symmetric diagonally dominant linear systems of equations. Boman and Hendrickson [5] were the first to discover the surprising relationship between these two seemingly unrelated problems. They applied the spanning trees of [2] to design solvers that run in time $m^{3/2}2^{O(\sqrt{\log n \log\log n})}\log(1/\epsilon)$. Spielman and Teng [14] improved their results by showing how to use the spanning trees of [2] to solve diagonally-dominant linear systems in time

$$m2^{O(\sqrt{\log n \log\log n})}\log(1/\epsilon).$$

By applying the low-stretch spanning trees developed in [9], the time for solving these linear systems reduces to

$$m\log^{O(1)} n \log(1/\epsilon),$$

and to $O(n(\log n \log\log n)^2 \log(1/\epsilon))$ when the systems are planar. Applying a recent reduction of Boman, Hendrickson and Vavasis [6], one obtains a $O(n(\log n \log\log n)^2 \log(1/\epsilon))$ time algorithm for solving the linear systems that arise when applying the finite element method to solve two-dimensional elliptic partial differential equations.

Recently Chekuri et al. [7] used low stretch spanning trees to devise an approximation algorithm for non-uniform buy-at-bulk network design problem. Their algorithm provides a first polylogarithmic approximation guarantee for this problem.

In another recent work Abraham et al. [1] use a techinique of star-decomposition introduced by Elkin et al. [9] to construct embeddings with a constant average stretch, where the average is over all *pairs of vertices*, rather than over all edges. The result of Abraham et al. [1] was, in turn, already used in a yet more recent work of Elkin et al. [10] on fundamental circuits.

Open Problems

The most evident open problem is to close the gap between the upper bound of $O(\log^2 n \log\log n)$ and the lower bound of $\Omega(\log n)$ on $avestr(n)$. Another intriguing subject is the study of low stretch spanning trees for various restricted families of graphs. Progress in this direction was recently achieved by Emek and Peleg [11] that constructed low stretch spanning trees with average stretch $O(\log n)$ for unweighted series-parallel graphs. Discovering other applications of low stretch spanning trees is another promising venue of study.

Finally, there is a closely related relaxed notion of low stretch *Steiner* or *Bartal* trees. Unlike a spanning tree, a Steiner tree does not have to be a subgraph of the original graph, but rather is allowed to use edges and vertices that were not present in the original graph. It is, however, required that the distances in the Steiner tree will be no smaller than the distances in the original graph. Low stretch Steiner trees were extensively studied [3,4,12]. Fakcharoenphol et al. [12] devised a construction of low stretch Steiner trees with an average stretch of $O(\log n)$. It is currently unknown whether the techniques used in the study of low stretch Steiner trees can help improving the bounds for the low stretch spanning trees.

Cross References

► Approximating Metric Spaces by Tree Metrics

Recommended Reading

1. Abraham, I., Bartal, Y., Neiman, O.: Embedding Metrics into Ultrametrics and Graphs into Spanning Trees with Constant Average Distortion. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms, New Orleans, January 2007
2. Alon, N., Karp, R.M., Peleg, D., West, D.: A graph-theoretic gane and its application to the *k*-server problem. SIAM J. Comput. **24**(1), 78–100 (1995). Also available Technical Report TR-91-066, ICSI, Berkeley (1991)
3. Bartal, Y.: Probabilistic approximation of metric spaces and its algorithmic applications. In: Proceedings of the 37th Annual Symposium on Foundations of Computer Science, Berlington, Oct. 1996 pp. 184–193
4. Bartal, Y.: On approximating arbitrary metrices by tree metrics. In: Proceedings of the 30th annual ACM symposium on Theory of computing, Dallas, 23–26 May 1998, pp. 161–168
5. Boman, E., Hendrickson, B.: On spanning tree preconditioners. Manuscript, Sandia National Lab. (2001)
6. Boman, E., Hendrickson, B., Vavasis, S.: Solving elliptic finite element systems in near-linear time with suppost preconditioners. Manuscript, Sandia National Lab. and Cornell, http://arXiv.org/abs/cs/0407022 Accessed 9 July 2004
7. Chekuri, C., Hagiahayi, M.T., Kortsarz, G., Salavatipour, M.: Approximation Algorithms for Non-Uniform Buy-at-Bulk Network Design. In: Proceedings of the 47th Annual Symp. on Foundations of Computer Science, Berkeley, Oct. 2006, pp. 677–686
8. Deo, N., Prabhu, G.M., Krishnamoorthy, M.S.: Algorithms for generating fundamental cycles in a graph. ACM Trans. Math. Softw. **8**, 26–42 (1982)
9. Elkin, M., Emek, Y., Spielman, D., Teng, S.-H.: Lower-Stretch Spanning Trees. In: Proc. of the 37th Annual ACM Symp. on Theory of Computing, STOC'05, Baltimore, May 2005, pp. 494–503
10. Elkin, M., Liebchen, C., Rizzi, R.: New Length Bounds for Cycle Bases. Inf. Proc. Lett. **104**(5), 186–193 (2007)
11. Emek, Y., Peleg, D.: A tight upper bound on the probabilistic embedding of series-parallel graphs. In: Proc. of Symp. on Discr. Algorithms, SODA'06, Miami, Jan. 2006, pp. 1045–1053
12. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. In: Proceedings of the 35th annual ACM symposium on Theory of Computing, San Diego, June 2003, pp. 448–455
13. Horton, J.D.: A Polynomial-time algorithm to find the shortest cycle basis of a graph. SIAM J. Comput. **16**(2), 358–366 (1987)
14. Spielman, D., Teng, S.-H.: Nearly-linear time algorithm for graph partitioning, graph sparsification, and solving linear systems. In: Proc. of the 36th Annual ACM Symp. on Theory of Computing, STOC'04, Chicago. USA, June 2004, pp. 81–90
15. Stepanec, G.F.: Basis systems of vector cycles with extremal properties in graphs. Uspekhi Mat. Nauk **19**, 171–175 (1964). (In Russian)
16. Zykov, A.A.: Theory of Finite Graphs. Nauka, Novosibirsk (1969). (In Russian)

LP Decoding

2002 and later; Feldman, Karger, Wainwright

Jonathan Feldman
Google, Inc., New York, NY, USA

Keywords and Synonyms

LP decoding; Error-correcting codes; Low-density parity-check codes; LDPC codes; Pseudocodewords; Belief propagation

Problem Definition

Error-correcting codes are fundamental tools used to transmit digital information over unreliable channels. Their study goes back to the work of Hamming and Shannon, who used them as the basis for the field of information theory. The problem of decoding the original information up to the full error-correcting potential of the system is often very complex, especially for modern codes that approach the theoretical limits of the communication channel.

LP decoding [4,5,8] refers to the application of *linear programming* (LP) *relaxation* to the problem of decoding an error-correcting code. Linear programming relaxation is a standard technique in approximation algorithms and operations research, and is central to the study of efficient algorithms to find good (albeit suboptimal) solutions to very difficult optimization problems [13]. LP decoders have tight combinatorial characterizations of decoding success that can be used to analyze error-correcting performance.

The codes for which LP decoding has received the most attention are *low-density parity-check* (LDPC) codes [9], due to their excellent error-correcting performance. The LP decoder is particularly attractive for analysis of these codes because the standard message-passing algorithms such as *belief propagation* (see [15]) used for decoding are often difficult to analyze, and indeed the performance of LP decoding is closely tied to these methods.

Error-Correcting Codes and Maximum-Likelihood Decoding

This section begins with a very brief overview of error-correcting codes, sufficient for formulating the LP decoder. Some terms are not defined for space reasons; for a full treatment of error-correcting codes in context, the reader is referred to textbooks on the subject (e. g., [11]).

A *binary error-correcting code* is a subset $C \subseteq \{0,1\}^n$. The *rate* of the code C is $r = \log(|C|)/n$. A *linear* binary code is a linear subspace of $\{0,1\}^n$. A *codeword* is a vector $y \in C$. Note that 0^n is always a codeword of a linear code, a fact that will be useful later. When the code is used for communication, a codeword $\dot{y} \in C$ is transmitted over a *noisy channel*, resulting in some *received word* $\hat{y} \in \Sigma^n$, where Σ is some alphabet that depends on the channel model. Generally in LP decoding a *memoryless, symmetric* channel is assumed. One common such channel is the *binary symmetric channel (BSC) with parameter* p, which will be referred to as BSC_p, where $0 < p < 1/2$. In the BSC_p, the alphabet is $\Sigma = \{0,1\}$, and for each i, the received symbol $\hat{y}_i$ is equal to $\dot{y}_i$ with probability p, and $\hat{y}_i = 1 - \dot{y}_i$ otherwise. Although LP decoding works with more general channels, this chapter will focus on the BSC_p.

The *maximum-likelihood (ML) decoding problem* is the following: given a received word $\hat{y} \in \{0,1\}^n$, find the codeword $y^* \in C$ that is most likely to have been sent over the channel. Defining the vector $\gamma \in \{-1,+1\}^n$ where $\gamma_i = 1 - 2\hat{y}_i$, it is easy to show:

$$y^* = \arg\min_{y \in C} \sum_i \gamma_i y_i \,. \tag{1}$$

The complexity of the ML decoding problem depends heavily on the code being used. For simple codes such as a *repetition code* $C = \{0^n, 1^n\}$, the task is easy. For more complex (and higher-rate) codes such as LDPC codes, ML decoding is NP-hard [1].

LP Decoding

Since ML decoding can be very hard in general, one turns to sub-optimal solutions that can be found efficiently. LP decoding, instead of trying to solve (1), relaxes the constraint $y \in C$, and instead requires that $y \in \mathcal{P}$ for some succinctly describable linear polytope $\mathcal{P} \subseteq [0,1]^n$, resulting in the following linear program:

$$y_{\mathrm{LP}} = \arg\min_{y \in \mathcal{P}} \sum_{i=1}^{n} \gamma_i y_i \,. \tag{2}$$

It should be the case that the polytope includes all the codewords, and does not include any integral non-codewords. As such, a polytope $\mathcal{P}$ is called *proper* for code C if $\mathcal{P} \cap \{0,1\}^n = C$.

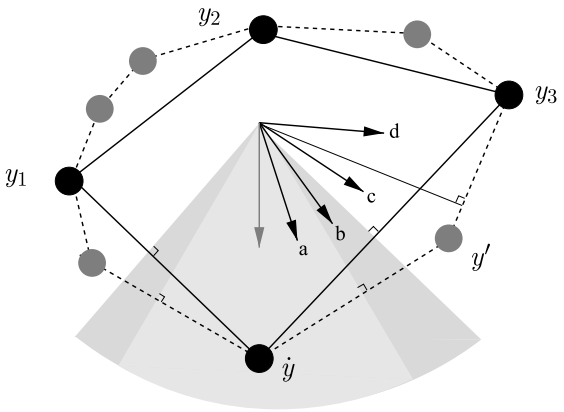
The LP decoder works as follows. Solve the LP in (2) to obtain $y_{\mathrm{LP}} \in [0,1]^n$. If y_{LP} is integral (i. e., all elements are 0 or 1), then output y_{LP}. Otherwise, output "error". By the definition of a proper polytope, if the LP decoder outputs a codeword, it is guaranteed to be equal to the ML codeword y^*. This fact is known as the *ML certificate* property.

Comparing with ML Decoding

A successful decoder is one that outputs the original codeword transmitted over the channel, and so the quality of an algorithm is measured by the likelihood that this happens.

(Another common non-probabilistic measure is the *worst-case* performance guarantee, which measures how many bit-flips an algorithm can tolerate and still be guaranteed to succeed.) Note that y^* is the one *most likely* to be the transmitted codeword $\dot{y}$, but it is not always the case that $y^* = \dot{y}$. However, no decoder can perform better than an ML decoder, and so it is useful to use ML decoding as a basis for comparison.

Figure 1 provides a geometric perspective of LP decoding, and its relation to exact ML decoding. Both decoders use the same LP objective function, but over different constraint sets. In exact ML decoding, the constraint set is the convex hull C of codewords (i. e., the set of points that are convex combinations of codewords from C), whereas relaxed LP decoding uses the larger polytope $\mathcal{P}$. In Fig. 1, the four arrows labeled (a)–(d) correspond to different "noisy" versions of the LP objective function. (a) If there is very little noise, then the objective function points to the transmitted codeword $\dot{y}$, and thus both ML decoding and LP decoding succeed, since both have the transmitted codeword $\dot{y}$ as the optimal point. (b) If more noise is introduced, then ML decoding succeeds, but LP decoding fails, since the fractional vertex y' is optimal for the relaxation. (c) With still more noise, ML decoding fails, since y_3 is now optimal; LP decoding still has a fractional optimum y', so this error is in some sense "detected". (d) Finally, with a lot of noise, both ML decoding and LP decoding have y_3 as the optimum, and so both methods fail and the error is "undetected". Note that in the last two cases (c, d), when ML decoding fails, the failure of the LP decoder is in some sense the fault of the code itself, as opposed to the decoder.



LP Decoding, Figure 1
A decoding polytope $\mathcal{P}$ (*dotted line*) and the convex hull C (*solid line*) of the codewords $\dot{y}$, y_1, y_2, and y_3. Also shown are the four possible cases (a–d) for the objective function, and the normal cones to both $\mathcal{P}$ and C

Normal Cones and C-Symmetry

The (negative) *normal cones* at $\dot{y}$ (also called the *fundamental cone* [10]) is defined as follows:

$$N_{\dot{y}}(\mathcal{P}) = \left\{\gamma \in \mathbb{R}^n : \sum_i \gamma_i(y_i - \dot{y}_i) \geq 0 \text{ for all } y \in \mathcal{P}\right\},$$

$$N_{\dot{y}}(C) = \left\{\gamma \in \mathbb{R}^n : \sum_i \gamma_i(y_i - \dot{y}_i) \geq 0 \text{ for all } y \in C\right\}.$$

Note that $N_{\dot{y}}(\mathcal{P})$ corresponds to the set of cost vectors γ such that $\dot{y}$ is an optimal solution to (2). The set $N_{\dot{y}}(C)$ has a similar interpretation as the set of cost vectors γ for which $\dot{y}$ is the ML codeword. Since $\mathcal{P} \subset C$, it is immediate from the definition that $N_y(C) \supset N_y(\mathcal{P})$ for all $y \in C$. Fig. 1 shows these two cones and their relationship.

The success probability of an LP decoder is equal to the total probability mass of $N_{\dot{y}}(\mathcal{P})$, under the distribution on cost vectors defined by the channel. The success probability of ML decoding is similarly related to the probability mass in the normal cone $N_y(C)$. Thus, the discrepancy between the normal cones of $\mathcal{P}$ and C is a measure of the gap between exact ML and relaxed LP decoding.

This analysis is specific to a particular transmitted codeword $\dot{y}$, but one would like to apply it in general. When dealing with linear codes, for most decoders one can usually assume that an arbitrary codeword is transmitted, since the decision region for decoding success is symmetric. The same holds true for LP decoding (see [4] for proof), as long as the polytope $\mathcal{P}$ is C-*symmetric*, defined as follows:

Definition 1 A proper polytope $\mathcal{P}$ for the binary code C is C-**symmetric** if, for all $y \in \mathcal{P}$ and $\dot{y} \in C$, it holds that $y' \in \mathcal{P}$, where $y'_i = |y_i - \dot{y}_i|$.

Using a Dual Witness to Prove Error Bounds

In order to prove that LP decoding succeeds, one must show that $\dot{y}$ is the optimal solution to the LP in (2). If the code C is linear, and the relaxation is proper and C-symmetric, one can assume that $\dot{y} = 0^n$, and then show that 0^n is optimal. Consider the *dual* of the decoding LP in (2). If there is a feasible point of the dual LP that has the same cost (i. e., zero) as the point 0^n has in the primal, then 0^n must be an optimal point of the decoding LP. Therefore, to prove that the LP decoder succeeds, it suffices to exhibit a zero-cost point in the dual.[1]

[1] Actually, since the existence of the zero-cost dual point only proves that 0^n is one of possibly many primal optima, one needs to be a bit more careful, a minor issue deferred to more complete treatments of this material.

Key Results

LP decoders have mainly been studied in the context of Low-Density Parity-Check codes [9], and their generalization to expander codes [12]. LP decoders for Turbo codes [2] have also been defined, but the results are not as strong. This summary of key results gives bounds on the *word error rate (WER)*, which is the probability, over the noise in the channel, that the decoder does not output the transmitted word. These bounds are relative to specific *families* of codes, which are defined as infinite set of codes of increasing length whose rate is bounded from below by some constant. Here the bounds are given in asymptotic form (without constants instantiated), and only for the binary symmetric channel.

Many other important results that are not listed here are known for LP decoding and related notions. Some of these general areas are surveyed in the next section, but there is insufficient space to reference most of them individually; the reader is referred to [3] for a thorough bibliography.

Low-Density Parity-Check Codes

The polytope $\mathcal{P}$ for LDPC codes, first defined in [4,8,10], is based on the underlying *Tanner graph* of the code, and has a linear number of variables and constraints. If the Tanner graph expands sufficiently, it is known that LP decoding can correct a constant fraction of errors in the channel, and thus has an inverse exponential error rate. This was proved using a dual witness:

Theorem 1 ([6]) *For any rate $r > 0$, there is a constant $\epsilon > 0$ such that there exists a rate r family of low-density parity-check codes with length n where the LP decoder succeeds as long as at most ϵn bits are flipped by the channel. This implies that there exists a constant $\epsilon' > 0$ such that the word error rate under the BSC_p with $p < \epsilon'$ is at most $2^{-\Omega(n)}$.*

Expander Codes

The *capacity* of a communication channel bounds from above the rate one can obtain from a family of codes and still get a word error rate that goes to zero as the code length increases. The notation C_p is used to denote the capacity of the BSC_p. Using a family of codes based on expanders [12], LP decoding can achieve rates that approach capacity. Compared to LDPC codes, however, this comes at the cost of increased decoding complexity, as the size of the LP is exponential in the gap between the rate and capacity.

Theorem 2 ([7]) *For any $p > 0$, and any rate $r < C_p$, there exists a rate r family of expander codes with length n such that the word error rate of LP decoding under the BSC_p is at most $2^{-\Omega(n)}$.*

Turbo Codes

Turbo codes [2] have the advantage that they can be encoded in linear time, even in a streaming fashion. *Repeat-accumulate* codes are a simple form of Turbo code. The LP decoder for Turbo codes and their variants was first defined in [4,5], and is based on the *trellis* structure of the component *convolutional* codes. Due to certain properties of turbo codes it is impossible to prove bounds for turbo codes as strong as the ones for LDPC codes, but the following is known:

Theorem 3 ([5]) *There exists a rate $1/2 - o(1)$ family of repeat-accumulate codes with length n, and a constant $\epsilon > 0$, such that under the BSC_p with $p < \epsilon$, the LP decoder has a word error rate of at most $n^{-\Omega(1)}$.*

Applications

The application of LP decoding that has received the most attention so far is for LDPC codes. The LP for this family of codes not only serves as an interesting alternative to more conventional iterative methods [15], but also gives a useful tool for analyzing those methods, an idea first explored in [8,10,14]. Iterative methods such as *belief propagation* use local computations on the Tanner graph to update approximations of the marginal probabilities of each code bit. In this type of analysis, the vertices of the polytope $\mathcal{P}$ are referred to as *pseudocodewords*, and tend to coincide with the fixed points of this iterative process. Other notions of pseudocodeword-like structures such as *stopping sets* are also known to coincide with these polytope vertices. Understanding these structures has also inspired the design of new codes for use with iterative and LP decoding. (See [3] for a more complete bibliography of this work).

The decoding method itself can be extended in many ways. By adding redundant information to the description of the code, one can derive tighter constraint sets to improve the error-correcting performance of the decoder, albeit at an increase in complexity. Adaptive algorithms that try to add constraints "on the fly" have also been explored, using branch-and-bound or other techniques. Also, LP decoding has inspired the use of other methods from optimization theory in decoding error-correcting codes. (Again, see [3] for references.)

Open Problems

The LP decoding method gives a simple, efficient and analytically tractable approach to decoding error-correcting codes. The results known to this point serve as a proof of concept that strong bounds are possible, but there are still important questions to answer. Although LP decoders can achieve capacity with decoding time polynomial in the length of the code, the complexity of the decoder still depends exponentially on the gap between the rate and capacity (as is the case for all other known provably efficient capacity-achieving decoders). Decreasing this dependence would be a major accomplishment, and perhaps LP decoding could help. Improving the fraction of errors correctable by LP decoding is also an important direction for further research.

Another interesting question is whether there exist constant-rate linear-distance code families for which one can formulate a polynomial-sized exact decoding LP. Put another way, is there a constant-rate linear-distance family of codes whose convex hulls have a polynomial number of facets? If so, then LP decoding would be equivalent to ML decoding for this family. If not, this is strong evidence that suboptimal decoding is inevitable when using good codes, which is a common belief.

An advantage to LP decoding is the *ML certificate* property mentioned earlier, which is not enjoyed by most other standard suboptimal decoders. This property opens up the possibility for a wide range of heuristics for improving decoding performance, some of which have been analyzed, but largely remain wide open.

LP decoding has (for the most part) only been explored for LDPC codes under memoryless symmetric channels. The LP for turbo codes has been defined, but the error bounds proved so far are not a satisfying explanation of the excellent performance observed in practice. Other codes and channels have gotten little, if any, attention.

Cross References

- ► Decoding Reed–Solomon Codes
- ► Learning Heavy Fourier Coefficients of Boolean Functions
- ► Linearity Testing/Testing Hadamard Codes
- ► List Decoding near Capacity: Folded RS Codes

Recommended Reading

1. Berlekamp, E., McEliece, R., van Tilborg, H.: On the inherent intractability of certain coding problems. IEEE Trans. Inf. Theory **24**, 384–386 (1978)
2. Berrou, C., Glavieux, A., Thitimajshima, P.: Near Shannon limit error-correcting coding and decoding: turbo-codes. In: Proc. IEEE Int. Conf. Comm. (ICC), pp. 1064–1070. Geneva, 23–26 May 1993
3. Boston, N., Ganesan, A., Koetter, R., Pazos, S., Vontobel, P.: Papers on pseudocodewords. HP Labs, Palo Alto. http://www.pseudocodewords.info.
4. Feldman, J.: Decoding Error-Correcting Codes via Linear Programming. Ph. D. thesis, Massachusetts Institute of Technology (2003)
5. Feldman, J., Karger, D.R.: Decoding turbo-like codes via linear programming. In: Proc. 43rd annual IEEE Symposium on Foundations of Computer Science (FOCS), Vancouver, 16–19 November 2002
6. Feldman, J., Malkin, T., Servedio, R.A., Stein, C., Wainwright, M.J.: LP decoding corrects a constant fraction of errors. In: Proc. IEEE International Symposium on Information Theory, Chicago, 27 June – 2 July 2004
7. Feldman, J., Stein, C.: LP decoding achieves capacity. In: Symposium on Discrete Algorithms (SODA '05), Vancouver, January (2005)
8. Feldman, J., Wainwright, M.J., Karger, D.R.: Using linear programming to decode linear codes. In: 37th annual Conf. on Information Sciences and Systems (CISS '03), Baltimore, 12–14 March 2003
9. Gallager, R.: Low-density parity-check codes. IRE Trans. Inform. Theory, IT-8 , pp. 21–28 (1962)
10. Koetter, R., Vontobel, P.: Graph covers and iterative decoding of finite-length codes. In: Proc. 3rd International Symposium on Turbo Codes and Related Topics, pp. 75–82, September 2003. Brest, France (2003)
11. MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error Correcting Codes. North-Holland, Amsterdam (1981)
12. Sipser, M., Spielman, D.: Expander codes. IEEE Trans. Inf. Theory **42**, 1710–1722 (1996)
13. Vazirani, V.V.: Approximation Algorithms. Springer, Berlin (2003)
14. Wainwright, M., Jordan, M.: Variational inference in graphical models: the view from the marginal polytope. In: Proc. 41st Allerton Conf. on Communications, Control, and Computing, Monticello, October (2003)
15. Wiberg, N.: Codes and Decoding on General Graphs, Ph. D. thesis, Linkoping University, Sweden (1996)